Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional Dalam Game *SuperTuxKart*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Arrizal Amin
NIM: 145150200111182

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional Untuk Bermain Game *SuperTuxKart*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Arrizal Amin
NIM: 145150200111182

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Yuita Arum Sari, S.Kom, M.Kom	Sigit Adinugroho, S.Kom., M.Sc
NIK: 2016098807152001	NIK: 2016078807011001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 8 Agustus 2018

Arrizal Amin

NIM: 145150200111182

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT karena atas segala rahmat dan limpahan hidayahNya, skripsi yang berjudul "Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional Untuk Bermain *Game SuperTuxKart*" ini dapat disusun dengan baik. Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada Program Studi Informatika / Ilmu Komputer, Universitas Brawijaya.

Pada kesempatan ini penulis mengucapkan banyak terimakasih atas segala bantuan dan dedikasi moral maupun material dalam rangka penyusunan skripsi ini. Atas bantuan yang diberikan, penulis mengucapkan banyak terimakasih kepada:

1. Kedua orang tua saya yang selalu memberikan dukungan dari segi manapun selama proses belajar di bangku perkuliahan Universitas Brawijaya, serta semua kerabat dan saudara yang telah mendoakan.
2. Yuita Arum Sari, S.Kom., M.Kom. selaku Dosen Pembimbing I yang telah meluangkan waktu untuk membimbing dan mengarahkan penulis dalam penulisan skripsi dengan penuh perhatian dan kesabaran.
3. Sigit Adinugroho, S.Kom., M.Sc. selaku Dosen Pembimbing II yang telah meluangkan waktu untuk membimbing dan mengarahkan penulis dalam penulisan skripsi dengan penuh perhatian dan kesabaran.
4. Seluruh Dosen Fakultas Ilmu Komputer, Universitas Brawijaya atas ilmu yang bermanfaat bagi penulis.
5. Teman-teman Informatika angkatan 2014 yang tidak dapat saya sebutkan satu persatu.

Penulis menyadari bahwa skripsi ini tentunya tidak terlepas dari berbagai kekurangan dan kesalahan. Oleh karena itu, segala kritik dan saran yang bersifat membangun sangat penulis harapkan dari berbagai pihak demi penyempurnaan penulisan skripsi ini.

Akhirnya penulis berharap agar skripsi ini dapat memberikan sumbangan dan manfaat bagi semua pihak yang berkepentingan.

Malang, 8 Agustus 2018

Arrizal Amin

arrizalamin@gmail.com

ABSTRAK

Arrizal Amin. 2017. Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional Dalam *Game SuperTuxKart*. Skripsi Program Studi Infomatika / Ilmu Komputer, Fakultas Ilmu Komputer Universitas Brawijaya.

Pembimbing: Yuita Arum Sari, S.Kom., M.Kom.

Salah satu komponen penting dari permainan video adalah kecerdasan buatan untuk membuat permainan menjadi lebih kompetitif. Kecerdasan buatan berperan dalam menentukan aksi yang akan dijalankan demi mencapai tujuan di dalam permainan. Dalam pembuatan algoritme kecerdasan buatan, pengembang permainan perlu memprogram kecerdasan buatan untuk dapat memilih aksi/langkah yang akan dipakai dalam setiap keadaan di dalam permainan yang memungkinkan. Dalam penelitian ini, jaringan saraf tiruan akan digunakan sebagai kecerdasan buatan dalam permainan video. Penggunaan jaringan saraf tiruan akan memudahkan pengembang karena tidak perlu memprogram algoritme kecerdasan buatan berdasarkan setiap keadaan yang ada. Selain itu jaringan saraf tiruan dapat beradaptasi dan dapat mempelajari perilaku pemain. Penelitian ini akan menggunakan contoh permainan SuperTuxKart untuk mengembangkan kecerdasan buatan. Kecerdasan buatan yang dibuat pada learning rate 0,0001, momentum 0,3 dan epoch ke-100 dapat mencapai akurasi 86,72% dalam meniru perilaku pemain saat memainkan permainan. Sehingga dapat disimpulkan bahwa penggunaan jaringan saraf tiruan dapat digunakan sebagai kecerdasan buatan dalam permainan video.

Kata kunci: jaringan saraf tiruan, jaringan saraf tiruan konvolusional, klon perilaku, kecerdasan buatan game

ABSTRACT

One of the important component of the video game is an artificial intelligence to make the game more competitive. Artificial intelligence used to decide action to reach goal in the game and challenge game player. In the process of developing artificial intelligence, developer needs to program an aritficial intelligence to make a decision for action for each states possible in the game. In this research, artifical neural network will be used as an artificial intelligence inside video game. Neural network will simplify process of developing artificial intelligence because developer does not have to program an algorithm to decide each action for each possible states in the game. Furthermore, neural network can learn or clone gamer's behavior while playing the game. In this research, SuperTuxKart will be used for an example to develop artificial intelligence inside video game. Artificial Intelligence with learning rate 0.0001, momentum 0.3 and epoch 100 reaches accuracy 86.72% for cloning game's behavior while playing video game. So this research concluded that neural network can be used as an artificial intelligence inside game.

Keywords: computer vision, artificial neural network, convolutional neural network, game artificial intelligence

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Game SuperTuxKart adalah jenis permainan balapan digital didalam komputer. Salah satu faktor penting didalam *game SuperTuxKart* adalah kecerdasan buatan yang ditanamkan didalamnya. Kecerdasan buatan membuat *game* menjadi kompetitif dan tidak membosankan. Kecerdasan buatan memiliki pengetahuan dalam lingkungan *game* untuk menentukan keputusan yang akan dilakukan dalam mencapai tujuan didalam *game*. Pengetahuan kecerdasan buatan dapat menentukan keputusan dengan aturan-aturan yang diberikan berdasarkan kondisi kecerdasan buatan pada setiap *frame*.

Game SuperTuxKart memiliki tingkat kesulitan untuk menyesuaikan dengan pemain. Tingkat kesulitan ini mempengaruhi kecerdasan buatan dalam menentukan aksinya atau rintangan pemain untuk mencapai tujuan. Dalam *game SuperTuxKart*, tingkat kesulitan memiliki properti kecerdasan buatan yang berbeda seperti kecepatan, nitro yang dimiliki dan probabilitas pada aksi yang akan dilakukan. Penentuan properti ini memerlukan pengujian terhadap pemain agar dapat dimainkan oleh seluruh pemain yang memiliki tingkatan yang bervariasi.

SuperTuxKart adalah *game* balapan yang bersumber terbuka. *Game* ini dipilih karena cukup ringan, sehingga *resource* yang tersisa dapat dimanfaatkan untuk jaringan syaraf tiruan untuk meniru perilaku pemain. Selain itu *game SuperTuxKart* juga sering dipakai sebagai objek penelitian jaringan saraf tiruan seperti klon perilaku pemain (Ross & Bagnell, 2010), analisis kebiasaan tangan yang digunakan pemain (Puzenat & Verlut, 2010) dan deteksi konsumsi alkohol pemain (Robinel & Puzenat, 2014).

Klon perilaku adalah teknik yang telah sukses pada berbagai masalah di dunia nyata, umumnya menggunakan algoritme *supervised machine learning* (Ross & Bagnell, 2010). Klon perilaku mencoba meniru perilaku manusia berdasarkan data yang telah direkam. Teknik ini lebih mudah diimplementasikan oleh pengembang *game* dan dapat memiliki kecerdasan yang dinamis terhadap tingkatan pemain. Penelitian sebelumnya menggunakan klon perilaku untuk membuat kecerdasan buatan dalam *game Street Fighter Zero 3 Upper* berdasarkan data pemain (Lueangrueangroj & Kotrajaras, 2012).

Penelitian ini akan menggunakan jaringan saraf tiruan konvolusional untuk melakukan klon perilaku pemain. Jaringan saraf tiruan konvolusional dipilih karena dapat mempelajari fitur pada citra secara otomatis sehingga dapat tidak perlu melakukan rekayasa fitur oleh manusia (Humphrey, et al., 2012). Penelitian ini akan menggunakan jaringan saraf tiruan dengan arsitektur *SqueezeNet* (Iandola, et al., 2016). *SqueezeNet* dipilih karena model ini memiliki teknik untuk mengkompresi parameter dengan mempertahankan akurasi. Sehingga proses pelatihan dan inferensi model ini lebih cepat dan dapat dipakai di komputer dan

GPU dengan memori yang terbatas. Data pemain yang dikumpulkan adalah tangkapan layar dan aksi yang sedang dilakukan.

Klon perilaku dalam menyederhanakan kompleksitas pembuatan sistem pakar yang berbasis kecerdasan buatan dalam *game* SuperTuxKart. Pengembang *game* tidak perlu membuat kecerdasan buatan dengan memprogram setiap keputusan yang akan dilakukan dalam setiap kondisi di dalam *game* agar kecerdasan buatan mampu bersaing dengan pemain. Berdasarkan paragraf di atas, maka penelitian ini akan mengambil judul Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional Dalam *Game SuperTuxKart*.

1.2 Rumusan masalah

Berdasarkan latar belakang, dapat diambil rumusan masalah sebagai berikut:

1. Bagaimana menerapkan jaringan saraf tiruan konvolusional untuk meniru perilaku pemain dalam bermain *SuperTuxKart*?
2. Bagaimana akurasi dalam menerapkan jaringan saraf tiruan konvolusional untuk meniru perilaku pemain dalam bermain *SuperTuxKart*?

1.3 Tujuan

Berdasarkan rumusan masalah, tujuan dari penelitian ini adalah sebagai berikut:

1. Menerapkan jaringan saraf tiruan konvolusional untuk meniru perilaku pemain dalam bermain *SuperTuxKart*.
2. Mengetahui akurasi dalam menerapkan jaringan saraf tiruan konvolusional untuk meniru perilaku pemain dalam bermain *SuperTuxKart.*

1.4 Manfaat

Mempermudah dalam membuat kecerdasan buatan di dalam *game* balapan dengan hanya memberi data pelatihan yang cukup dan benar. Pengembang kecerdasan buatan dalam *game* tidak perlu memprogram setiap keputusan untuk setiap kondisi yang ada di dalam permainan. Selain itu, kecerdasan buatan dapat menyesuaikan diri dengan tingkatan pemain dengan belajar dari perilaku pemain dalam memainkan *game*.

1.5 Batasan masalah

Berdasarkan permasalahan akan diberikan batasan masalah sebagai berikut:

1. Data yang diambil dari pemain adalah data tangkapan layar dan aksi ketika pemain melakukan akselerasi, belok kanan, belok kiri, dan menggunakan *nitro*.
2. Data diambil saat pemain sedang memainkan *game*.
3. Data berjumlah 10.000 pada masing-masing label.

4. Pada penelitian ini hanya melakukan pengujian dan pelatihan pada mode *game time trial*.

1.6 Sistematika pembahasan

Skripsi ini disusun berdasarkan sistematika penulisan sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisi tentang latar belakang, rumusan msalah, batasan masalah, tujuan penelitian, manfaat penelitian dan sistematika penulisan.

BAB II KAJIAN KEPUSTAKAAN DAN DASAR TEORI

Bab ini berisi penelitian-penelitian sebelumnya dan teori-teori yang berhubungan dengan klon perilaku, jaringan saraf tiruan konvolusional dan *game SuperTuxKart*.

BAB III METODE PENELITIAN DAN PERANCANGAN

Bab ini menjelaskan mengenai klon perilaku dan jaringan saraf tiruan konvolusional.

BAB IV IMPLEMENTASI

Bab ini menjelaskan mengenai analisis kebutuhan dan perancangan sistem untuk penentuan jumlah kendaraan menggunakan pengembangan jaringan saraf tiruan konvolusional untuk melakukan klon perilaku pemain *game SuperTuxKart*.

BAB V PENGUJIAN DAN ANALISIS

Bab ini berisi hasil implementasi ke perangkat lunak, uji coba sistem dan analisis hasil.

BAB VI PENUTUP

Bab ini berisi kesimpulan berdasarkan penelitian yang telah dilakukan, serta saran-saran untuk pengembangan peneliti lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Klon Perilaku

Klon perilaku adalah teknik untuk melatih kecerdasan buatan untuk meniru perilaku pakar berdasarkan data yang diberikan. Klon perilaku biasanya diimplementasikan dan diuji pada lingkungan simulasi menggunakan subyek non-pakar (Abbott, 2007). Pada penelitiannya Abbot mencoba melakukan teknik klon perilaku pada agen untuk bermain RoboCup menggunakan rekaman data saat subyek bermain.

Klon perilaku juga dipakai dalam model AlphaGo untuk bermain permainan Go mengalahkan Grand Master (Silver, et al., 2016) dan mengalahkan Grand Master Dota 2 (OpenAI, 2017). Dari dunia *game* telah ada penelitian untuk melatih agen memainkan *game* seperti *Mario Tennis*, *Street Fighter Zero 3* dan *Super Smash Bros* menggunakan klon perilaku (Ross & Bagnell, 2010) (Lueangrueangroj & Kotrajaras, 2012) (Chen & Yi, 2017).

2.2 Jaringan Saraf Tiruan Konvolusional

Jaringan saraf tiruan konvolusional adalah jenis jaringan saraf tiruan dimana bidang reseptif (neuron) adalah unit konvolusi yang diberikan bobot dan digeser satu per satu terhadap seluruh array 2 dimensi dari *input*. Hasil dari lapisan konvolusi adalah array 2 dimensi yang dinamakan *activation map* yang akan menjadi *input* ke lapisan selanjutnya (Schmidhuber, 2014).

Jaringan saraf tiruan konvolusi terinspirasi oleh penelitian Hubel dan Wiesel mengenai sistem saraf visual pada kucing dan bagaimana otak merespon terhadap stimulasi visual. Dari penelitian tersebut disimpulkan bahwa masing-masing neuron hanya aktif pada sebagian kecil pada bidang visual dan aktif hanya pada orientasi tertentu. Kemudian disimpulkan bahwa sistem saraf visual terdiri dari 2 macam sel, yaitu *Simple-cell* (*S-cell*) dan *Complex-cell* (*C-cell*).

Kemudian penelitian tersebut diimplementasikan kedalam *Neocognitron* (Fukushima, 1980). Neocognitron adalah jaringan saraf tiruan yang terinspirasi dari penemuan Hubel & Wiesel tentang hirarki model dari sistem saraf visual. Neocognitron berhasil mengenali berbagai macam pola sederhana, huruf, dan angka, tetapi Neocognitron tidak menggunakan bobot yang dilatih melalui *Backpropagation* melainkan menggunakan *self-organizing network*.

2.2.1 *Rectified Linear Unit* (ReLU)

ReLU adalah salah satu fungsi aktivasi yang populer dalam jaringan saraf tiruan. Pertama kali diperkenalkan oleh Nair dan memiliki akurasi yang lebih baik dibanding *binary stepped function* (Nair & Hinton, 2009). Komputasi ReLU lebih cepat dibandingkan dengan sigmoid dan tanh. Selain itu ReLU mencegah masalah *vanishing gradient* saat propagasi mundur.

$$ReLU(x) = \max(x, 0) \quad (2.1)$$

Pada persamaan 2.1 fungsi relu hanya membandingkan input dengan nilai 0, jika input negatif maka akan diganti nilai 0.

2.2.2 Pooling

Lapisan *pooling* adalah lapisan yang melakukan *downsampling* terhadap *input* untuk mengecilkan dimensi. Lapisan *pooling* memiliki filter 2 dimensi yang digeser satu per satu terhadap seluruh array 2 dimensi dari *input* (*activation map*) (Weng, et al., 1992). Masing-masing filter memiliki *N*x*N input* dan 1 *output* yaitu nilai maksimum. Lapisan *pooling* juga terinspirasi dari model *Neocognitron*. Tetapi *Neocognitron* menggunakan metode *Winner-Take-All* (WTA) untuk melakukan subsampling (Schmidhuber, 2014).

2.3 Dropout Regularization

Dropout adalah teknik dimana beberapa neuron dalam jaringan saraf tiruan dinonaktifkan selama proses pelatihan untuk meningkatkan generalisasi dan membantu mencegah *overfitting*. *Overfitting* dapat dicegah dengan memaksa masing-masing neuron mempelajari fitur yang secara general berguna untuk menentukan jawaban yang benar (Hinton, et al., 2012). Selain itu *Dropout* juga bisa dilihat sebagai cara melatih banyak model jaringan saraf tiruan yang disatukan (Baldi & Sadowski, 2014).

2.4 Gradient Descent

Gradient Descent adalah algoritme paling populer untuk melakukan optimisasi dan paling umum untuk melatih jaringan saraf tiruan. *Gradient Descent* adalah cara untuk meminimalkan keluaran sebuah fungsi yang memiliki parameter dan mengubah secara bertahap parameter fungsi ke arah berlawanan dari gradien fungsi. Jumlah perubahan setiap tahapnya ditentukan oleh parameter *learning rate* (Ruder, 2016).

2.4.1 Backpropagation

Backpropagation adalah nama lain dari modus berbalik pada differensiasi otomatis (Werbos, 2006). Differensiasi otomatis adalah algoritme untuk mengevaluasi turunan dari fungsi, transformasi tersebut berdasarkan rumus turunan pada operasi aritmatika dan berbagai fungsi intrinsik yang dapat diturunkan yang terdiri dari beberapa langkah algoritme. Modus berbalik pada differensiasi otomatis menggunakan hasil akhir dari fungsi kemudian menggunakan aturan rantai sampai ke masing-masing variabel independen (Rall, 2006).

Backpropagation digunakan dalam jaringan saraf tiruan untuk meminimalkan error menggunakan algoritme *gradient descent*. Penelitian mengenai Backpropagation pertama kali digunakan pada fungsi untuk mencari nilai

keputusan yang optimal (Dreyfus, 1962). Kemudian pada tahun 1986, *Backpropagation* digunakan untuk melatih *Multi Layer Perceptron* (Rumelhart, Hinton, dan Williams. 1986). Pada penelitian tersebut Rumelhart et. al. menunjukkan hasil dari penyesuaian bobot, *hidden layer* yang mana bukan bagian dari *input* atau output, turut berperan untuk merepresentasikan fitur-fitur penting dan keteraturan dalam tugas yang diberikan yang ditangkap oleh interaksi masing-masing neuron.

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.2}$$

$$x \leftarrow x - \alpha \frac{\partial f(x)}{\partial x} \tag{2.3}$$

Pada persamaaan 2.2 adalah mencari gradien dengan menghitung turunan parsial suatu fungsi f terhadap variabel x (bobot). Persamaan 2.3 menghitung besar langkah ke arah titik minimum lokal dengan mengalikannya dengan *learning rate* (α) dan modifikasi bobot dengan mengurangkannya dengan besar langkah.

2.4.2 *Adaptive Moment Estimation* (*Adam*)

Adam adalah algoritme optimasi *gradient descent* yang menggabungkan 2 algoritme optimasi sebelumnya yaitu *Momentum* dan *RMSprop*. Algoritme ini melakukan *update* pada percepatan rata-rata gradien (*m*) atau momen pertama dan gradien kuadrat (*v*) atau momen kedua, pada *epoch* sebelumnya. Algoritme ini memiliki tambahan 2 *hyper-parameter* selain *learning rate*. β_1 adalah parameter yang mengontrol perlambatan dari percepatan rata-rata gradien. Sedangkan β_2 adalah parameter yang mengontrol pengurangan dari gradien kuadrat dan jumlah gradien kuadrat baru yang akan diakumulasi (Kingma & Ba, 2014).

Momen pertama dalam algoritme ini bertujuan mempercepat konvergensi ke arah minimum lokal berdasarkan percepatan rata-rata gradien sebelumnya. Momen pertama juga menghindari masalah ketika jalur menuju minimum lokal yang optimal berbukit-bukit sehingga membuat alur optimisasi menjadi zig-zag dan konvergensi dini. Momen kedua bertujuan mengadaptasi *learning rate* terhadap parameter dimana melakukan *update* kecil untuk parameter di*update* dan *update* yang lebih besar untuk parameter yang jarang (Duchi, et al., 2011). Momen kedua ini sangat baik untuk digunakan dalam keadaan data yang tersebar dan pelatihan yang *stochastic* atau *minibatch*, dimana arah ke minimum lokal akan lebih halus dan tidak zig-zag.

Selain itu algoritme ini juga memiliki tahap *bias-correction* dimana gradien akan mendekati sangat kecil beberapa langkah pertama dimana percepatan rata-rata dan gradien kuadrat adalah 0.

$$g = \nabla f(\theta_{t-1}) \tag{2.4}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g \tag{2.5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g^2 \tag{2.6}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.7}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.8}$$

$$\theta_t = \frac{\theta_{t-1} - \alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{2.9}$$

Dalam persamaan 2.4, gradien dari fungsi dihitung. Kemudian dilanjutkan dengan memperbarui momentum (*m*), dimana *m* lama dan gradien diperkecil oleh *hyperparameter* β_1. Nilai β_1 untuk mengatur besarnya momentum yang digunakan saat melakukan proses *update* bobot. Hal yang sama juga dilakukan untuk momentum kedua (*v*), momentum kedua menyimpan total gradient kuadrat pada masing-masing bobot untuk mengatur *learning rate* pada setiap bobot. β_2 dingunakan untuk mengatur jumlah gradien kuadrat yang akan dijumlahkan. Kemudian pada persamaan 2.7 dan 2.8 dilakukan koreksi bias, β_1^t dan β_2^t adalah *hyper-parameter* beta yang dipangkatkan dengan *epoch*. Terakhir pada persamaan 2.9, dilakukan pembaruan parameter.

2.5 SqueezeNet

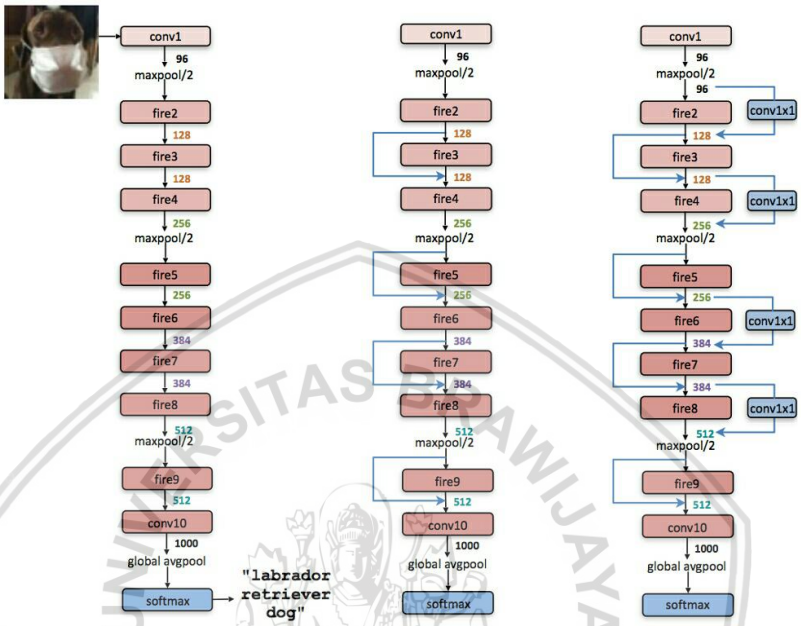
SqueezeNet merupakan arsitektur jaringan saraf tiruan yang menggunakan jaringan saraf tiruan konvolusional. SqueezeNet mampu mencapai akurasi *AlexNet* (pemenang *ImageNet classification task* 2012) dengan parameter 50 kali lebih sedikit dan waktu pelatihan 2 kali lebih cepat (Iandola, et al., 2016).

SqueezeNet banyak mengganti lapisan *convolution* 3x3 dengan 1x1 dan filter yang lebih sedikit untuk mengecilkan dimensi *activation map* (*squeeze*). Teknik reduksi dimensi ini pertama kali dikenalkan dalam model *Network In Network* (NIN) (Lin, et al., 2013). Selain itu lapisan *convolution* 1x1 atau disebut *Cross Channel Information Learning* secara biologis terinspirasi dari Visual Cortex manusia yang memiliki bidang reseptif (filter) yang dapat diatur ke orientasi yang berbeda-beda.

Proses *squeeze* dilanjutkan dengan lapisan *convolution* dengan filter lebih banyak untuk memperbesar kembali *activation map* (*Expand*).

Pada Gambar 2.1, adalah 3 macam arsitektur *SqueezeNet* yang dipaparkan oleh pembuat arsitekturnya (Iandola, et al., 2016). Pada arsitektur pertama menggunakan *fire module* pada model untuk melakukan klasifikasi anjing. Pada arsitektur kedua dan ketiga menggabungkan *fire module* dengan *residual learning* yang terinspirasi dari *ResNet* (He, et al., 2015).

Selain itu SqueezeNet juga mengganti lapisan *multi layer perceptron* dengan lapisan *squeeze* dan *Global Average Pool*. *Global Average Pool* mengambil rata-rata dari lapisan *convolution* terakhir untuk menentukan prediksi. Selain itu *multi-layer perceptron* lebih cenderung *overfitting* dan sangat bergantung pada regularisasi dropout. Sedangkan *Global Average Pool* karena tidak memiliki parameter sehingga *overfitting* dapat dihindari pada lapisan ini (Lin, et al., 2013).



Gambar 2.1 Arsitektur SqueezeNet (Iandola, et al., 2016)

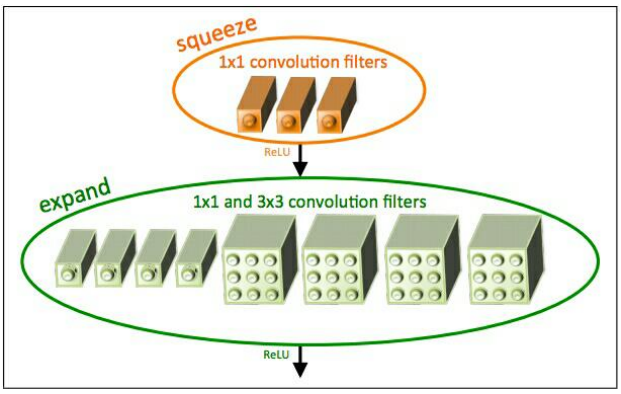
2.5.1 Fire Module

Fire module adalah lapisan *convolution* yang terdiri dari 2 lapisan yaitu lapisan *squeeze* dan *expand*. Lapisan *squeeze* melakukan reduksi dimensi dan mempelajari hubungan antar *activation map*. Sedangkan lapisan *expand* melakukan ekspansi dimensi dan mempelajari pola-pola *activation map* yang ada pada lapisan *squeeze* (Iandola, et al., 2016).

2.6 SuperTuxKart

SuperTuxKart adalah permainan balapan 3D gratis dan sumber terbuka. Permainan ini dapat dimainkan lebih dari 4 orang atau melawan kecerdasan buatan bawaan permainan. Permainan ini merupakan proyek lanjutan dari permainan TuxKart yang sudah tidak dilanjutkan pengembangannya (SuperTuxKart Team, 2016).

Permainan SuperTuxKart dapat dimainkan di berbagai sistem operasi yaitu Linux, Windows, dan MacOS. Selain itu permainan ini juga dapat dimainkan menggunakan *keyboard*, *joystick*, dan *wiimote*.



Gambar 2.2 Arsitektur *Fire Module* (Iandola, et al., 2016)

2.7 *Cross Entropy*

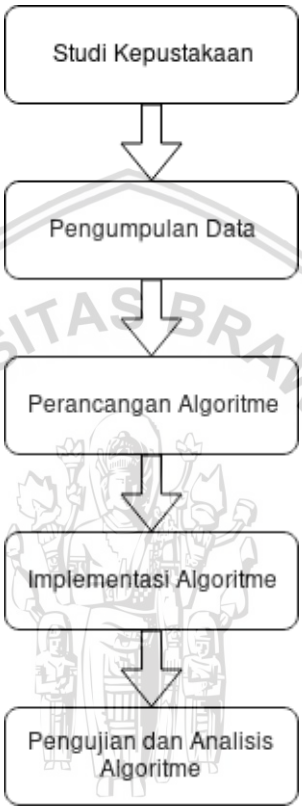
Cross entropy didefinisikan untuk distribusi probabilitas diskrit *D,* $\overline{D}$. Cross entropy mengukur seberapa banyak informasi yang dibutuhkan rata-rata untuk mengidentifikasi sampel dari $\overline{D}$ ketika menggunakan skema pengkodean untuk *D* (Green, 2016). Dalam jaringan saraf tiruan, fungsi *error cross entropy* dapat menghilangkan masalah pengurangan kecepatan proses pelatihan ketika menggunakan *squared error* ketika nilai prediksi jaringan saraf tiruan mendekati 1 (Nielsen, 2015).

$$xe(y', y) = -\sum_{i=1}^{Y} \ln(y_i')\, y_i \tag{2.10}$$

Berdasarkan persamaan 2.9, *cross entropy* dihitung dengan mengalikan ln(*y'*) dimana *y'* adalah prediksi jaringan saraf tiruan dan *y* yaitu label data latih pada setiap label. Penjumlahan hasil perkalian pada seluruh label kemudian dikali -1 untuk membuat nilai menjadi positif.

BAB 3 METODOLOGI

Bab 3 ini berisi langkah-langkah dalam penelitian yaitu studi kepustakaan, pengumpulan data, perancangan algoritme, implementasi algoritme, pengujian dan analisis algoritme. Kesimpulan dan saran juga dituliskan sebagai catatan untuk penelitian yang serupa dimasa mendatang. Gambar 3.1 merupakan diagram alur dalam penelitian:



Gambar 3.1 Diagram alir metodologi penelitian

3.1 Studi Pustaka

Tahapan pertama dalam bab ini adalah studi kepustakaan. Studi kepustakaan adalah proses pengumpulan sumber-sumber penelitian yang berhubungan dengan topik penelitian ini dan akan menunjang dalam kesuksesan penelitian ini. Jenis penelitian ini adalah non implementatif-analitik. Studi kepustakaan dapat berfungsi untuk memperdalam pengetahuan dan teori yang akan digunakan dalam penelitian ini. Studi kepustakaan yang digunakan dalam penelitian ini adalah sebagai berikut:

- Klon perilaku
- Jaringan saraf tiruan konvolusional
- *Pooling*
- Fungsi aktifasi ReLU

- *Backpropagation*
- *SuperTuxKart*

3.2 Pengumpulan Data

Tahapan kedua adalah proses pengumpulan data. Data yang dikumpulkan akan digunakan sebagai data latih dan uji. Data yang akan diambil pada tahap ini berupa tangkapan layar beserta aksi yang dijalankan oleh pemain. Data didapatkan dengan cara memainkan *game* ketika program pengumpul data berjalan. Fitur yang ada pada data hanya gambar tangkapan layar beserta label aksi yang dijalankan. Tahap pengumpulan dan pemrosesan data akan dijelaskan pada subbab 4.2.

3.3 Perancangan Algoritme

Tahapan ke tiga adalah menjelasakan perancangan algoritme yang mampu menyelesaikan permasalahan pada penelitian yang diangkat penulis. Dalam perancangan sistem dilakukan pembuatan arsitektur *SqueezeNet* dengan melalui proses pelatihan dan proses prediksi.

3.4 Implementasi Algoritme

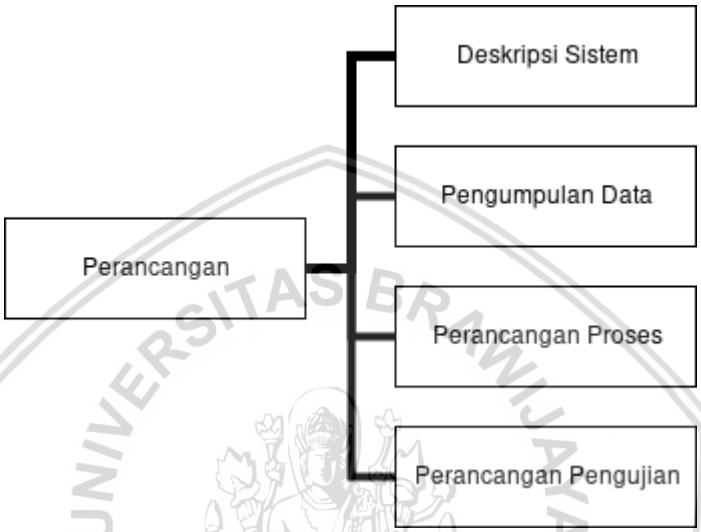
Tahapan keempat adalah mengimplementasikan algoritme sesuai dengan rancangan yang telah dibuat. Proses implementasi yaitu membangun algoritme yang telah dirancang ke dalam bahasa pemrograman. Sistem yang akan dibuat akan menerapkan metode yang dipilih untuk menyelesaikan permasalahan yang diangkat Implementasi system diterapkan dengan bahasa pemrograman Python 3.6 dengan memanfaatkan beberapa *library* yang menunjang pengerjaan yaitu *numpy*, *matplotlib*, *opencv*, *tqdm*, *mss*, dan *keyboard*.

3.5 Pelatihan dan Pengujian Algoritme

Tahapan kelima adalah pengujian jaringan saraf tiruan konvolusional yang telah implementasi. Pelatihan jaringan saraf tiruan menggunakan data berupa tangkapan layar yang telah diambil beserta label tombol *keyboard* yang ditekan. Kemudian dilakukan pengujian jaringan saraf tiruan untuk mengetahui performa algoritme yang telah dibuat berdasarkan data dan label yang ada untuk mengamati hasil akurasi jaringan saraf tiruan. Pengujian menggunakan nilai *error* dan akurasi dalam meniru perilaku pemain. Pengujian menggunakan 3 variasi yaitu: Pengaruh *epoch*, Pengaruh *learning rate* dan Pengaruh *momentum*.

BAB 4 PERANCANGAN

Pada Bab ini penelitian akan difokuskan pada tahap perancangan yang diimplementasikan pada sistem "Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional Dalam *Game SuperTuxKart*" dimana perancangannya meliputi deskripsi sistem, persiapan data, perancangan algoritme jaringan saraf tiruan perancangan pengujian. Gambar 4.1 adalah diagram perancangan sistem dalam penelitian ini:



Gambar 4.1 Diagram Perancangan Sistem

4.1 Deskripsi Algoritme

Algoritme yang dibuat pada penelitian ini adalah algoritme yang menggunakan pemrograman bahasa pemrograman Python dalam implementasinya. Tujuan utama algoritme adalah untuk menentukan aksi pada *game SuperTuxKart* berdasarkan *input* tangkapan layar yang diberikan. Untuk kontribusi, penelitian ini akan menguji teori penggunaan metode jaringan saraf tiruan dengan model *SqueezeNet* dan algoritme *Adam* dalam proses klon perilaku. Manfaat utama dari penelitian ini adalah sebagai metode alternatif bagi pengembang *game* untuk membuat algoritme kecerdasan buatan yang *general* dan dapat menyesuaikan kemampuan pemain.

4.2 Pengumpulan Data

Tahapan pengumpulan data dalam penelitian ini adalah proses mengumpulkan data tangkapan layar *game* beserta aksi yang dijalankan selama bermain game SuperTuxKart.

Dari proses pengumpulan data ini diambil masing-masing 10000 sampel gambar dari masing-masing aksi pada *game*. Jumlah data dari seluruh aksi pada *game* adalah 40.000.

Gambar 4.2 Pengumpulan Data

Proses pengumpulan data dilakukan seperti Gambar 4.2, dimana pemain akan memainkan *game* seperti biasa, dan program pengumpul data akan mencatat aksi yang dijalankan beserta tangkapan layar. Berdasarkan Gambar 4.2, label terdiri dari *array* sebesar 4 elemen. Masing-masing indeks mewakili aksi atau tombol *keyboar*d yang dilakukan pemain, 1 menujukkan aksi sedang dijalankan pada tangkapan layar, 0 menunjukkan tidak dijalankan.

4.3 Perancangan Arsitektur Jaringan Saraf Tiruan

Tahap perancangan arsitektur dalam penelitian ini perancangan jaringan saraf tiruan yang akan digunakan dalam klon perilaku. Arsitektur yang digunakan dalam penelitian ini adalah modifikasi dari arsitektur *SqueezeNet* (Iandola, et al., 2016). Karena label data yang lebih sedikit, dan ukuran *input* yang lebih kecil maka parameter lapisan *convolution* dan *fire module* diperkecil 4 kali lipat. Selain itu *filter max pooling* 2 dan 3 juga diperkecil karena ukuran *input* yang lebih kecil menjadi 2. Arsitektur lengkap jaringan saraf tiruan disajikan dalam Tabel 4.1.

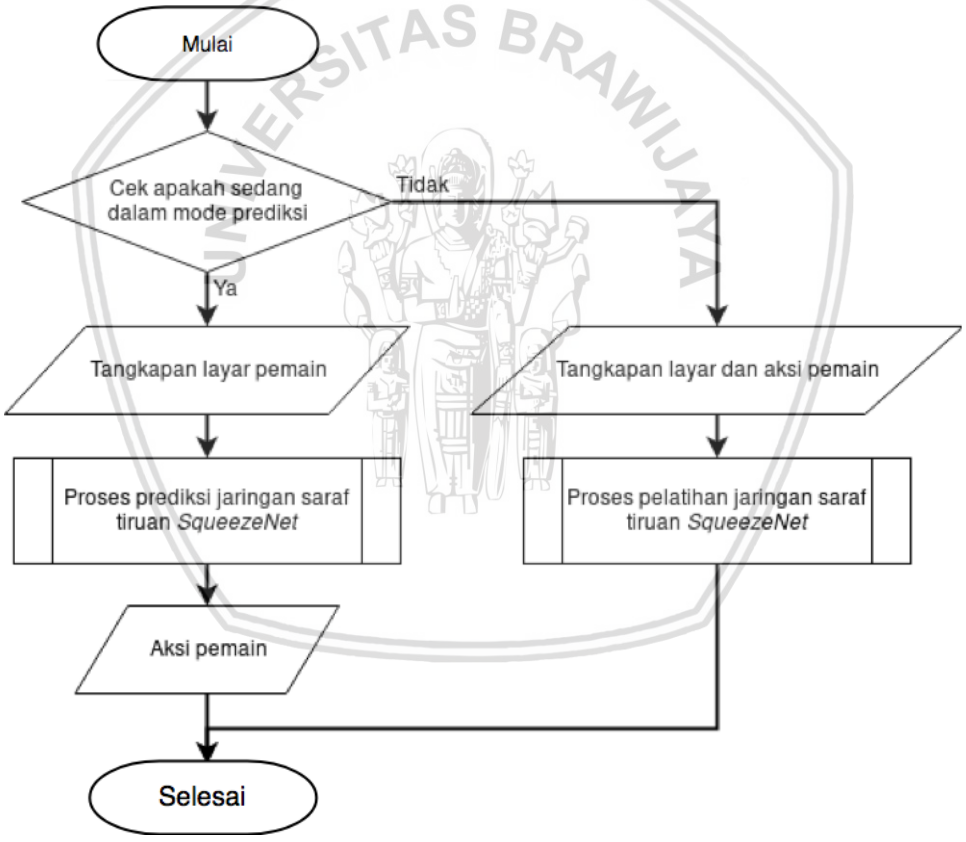
Tabel 4.1 Arsitektur jaringan saraf tiruan

Nama	Jumlah Filter	Ukuran Filter	Stride	Squeeze	Expand
conv_input	16	3x3	2	-	-
relu_input	-	-	-	-	-
max_pool_1	-	3x3	2	-	-
fire_1	-	-	-	4	16
fire_2	-	-	-	4	16
max_pool_2	-	2x2	2	-	-
fire_3	-	-	-	8	32
fire_4	-	-	-	8	32
max_pool_2	-	2x2	2	-	-
fire_5	-	-	-	12	48
fire_6	-	-	-	12	48
fire_7	-	-	-	16	64

dropout(0,5)	-	-	-	-	-
conv_out	4	1x1	1	-	-
relu_out	-	-	-	-	-
avg_pool	-	-	-	-	-

4.4 Perancangan Proses

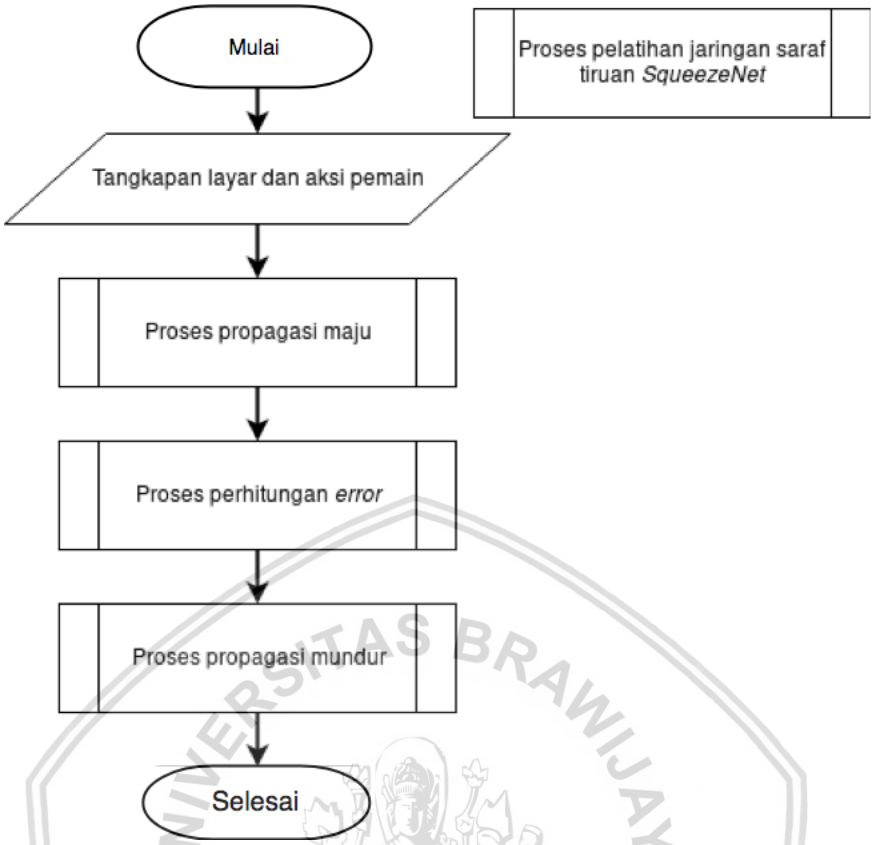
Pada perancangan proses ini akan menjelaskan proses klon perilaku dengan memasukkan tangkapan layar beserta aksi pemain yang kemudian akan dimasukkan kedalam jaringan saraf tiruan. Model jaringan saraf tiruan terdiri dari 2 mode yaitu pelatihan dan prediksi. Dalam proses pelatihan masukan ke dalam sistem adalah tangkapan layar dan aksi pemain, sedangkan dalam proses prediksi hanya tangkapan layar saja. Proses dari sistem klon perilaku disajikan dalam diagram alir Gambar 4.3.



Gambar 4.3 Diagram Alir Sistem

4.4.1 Proses Pelatihan Jaringan Saraf Tiruan Konvolusional

Proses pelatihan akan digunakan untuk melatih jaringan saraf tiruan untuk mempelajari pola di dalam *dataset* dan hubungan antar data dan labelnya dengan memperkecil *error* dalam propagasi mundur. Diagram alir dari algoritme dapat dilihat pada Gambar 4.4.



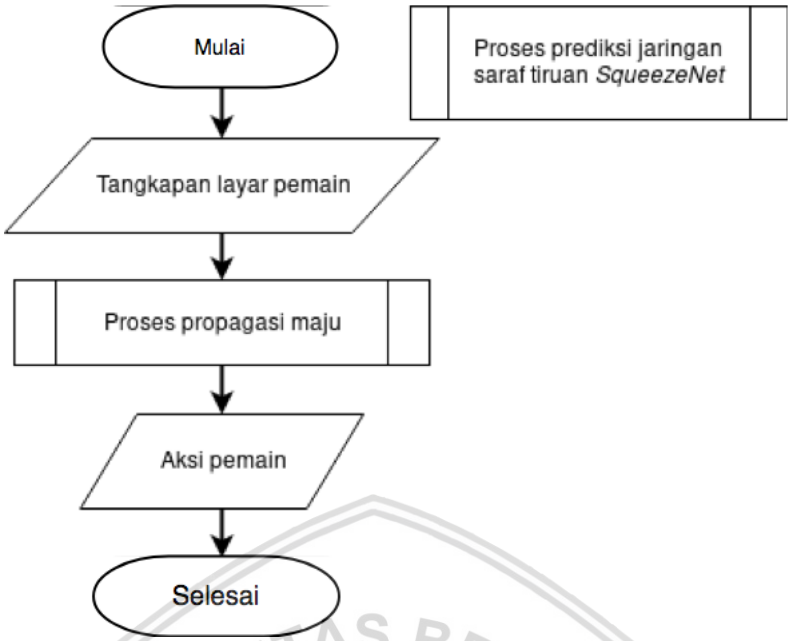
Gambar 4.4 Proses pelatihan jaringan saraf tiruan konvolusional

Diagram alir dalam Gambar 4.4, algoritme menerima *input* berupa tangkapan layar dan aksi pemain yang dilakukan. Kemudian *input* diteruskan ke proses propagasi maju untuk melakukan prediksi. Prediksi yang didapat dihitung nilai *error*nya pada proses perhitungan *error*. Proses perhitungan error memiliki *output* nilai *error* dan turunan parsial terhadap *input* yang akan digunakan pada proses selanjutnya. Turunan parsial yang didapat digunakan pada proses propagasi mundur untuk menghitung gradien dari keseluruhan model terhadap bobot jaringan saraf tiruan.

4.4.2 Proses Prediksi Jaringan Saraf Tiruan Konvolusional

Proses prediksi lebih sederhana daripada proses pelatihan jaringan saraf tiruan. Proses ini digunakan untuk melakukan uji coba pada jaringan saraf tiruan yang telah dilatih. Diagram alir dari algoritme dapat dilihat pada Gambar 4.5.

Diagram alir dalam Gambar 4.5, algoritme dimulai dengan *input* yang berupa tangkapan layar pemain. *Input* kemudian teruskan ke proses propagasi maju untuk mendapatkan *output* berupa aksi pemain.



Gambar 4.5 Proses prediksi jaringan saraf tiruan konvolusional

4.4.3 Proses Propagasi Maju

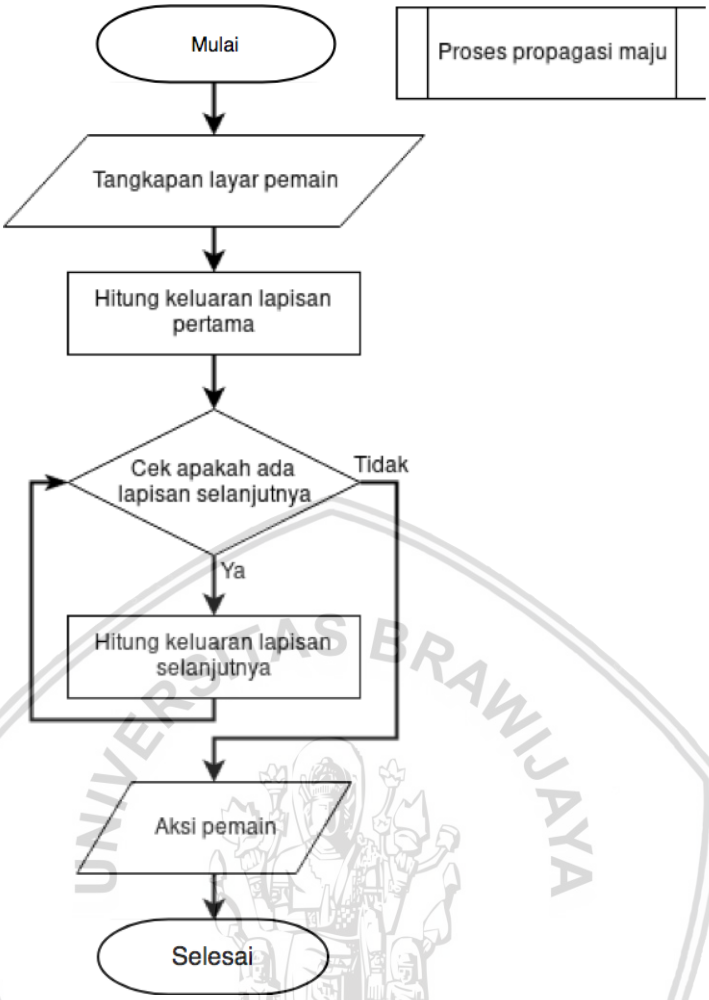
Proses propagasi maju adalah algoritme yang memasukkan *input* ke dalam setiap lapisan jaringan saraf tiruan untuk mendapatkan prediksi. Diagram alir dari algoritme dapat dilihat di Gambar 4.6.

Diagram alir dalam Gambar 4.6, algoritme menerima *input* berupa tangkapan layar pemain. Input kemudian dimasukkan ke dalam setiap lapisan dari jaringan saraf tiruan hingga lapisan terakhir. *Output* dari lapisan terakhir berupa aksi pemain yang diprediksi.

4.4.4 Proses Perhitungan *Error*

Proses perhitungan *error* membandingkan prediksi jaringan saraf tiruan dengan label data sebernarnya. Memperkecil *Error* merupakan tujuan utama jaringan saraf tiruan agar memiliki akurasi yang lebih baik. Oleh karena ini lapisan ini adalah lapisan pertama dalam propagasi mundur yang menghitung turunan parsial terhadap *input*. Diagram alir dari algoritme dapat dilihat di Gambar 4.7.

Berdasarkan diagram alir pada Gambar 4.7, algoritme menerima *input* berupa prediksi dan label. Kedua *input* kemudian dihitung nilai *error*nya beserta turunan parsial terhadap prediksinya. *Output* dari algoritme ini adalah turunan parsial terhadap prediksi dan nilai error jaringan saraf tiruan.

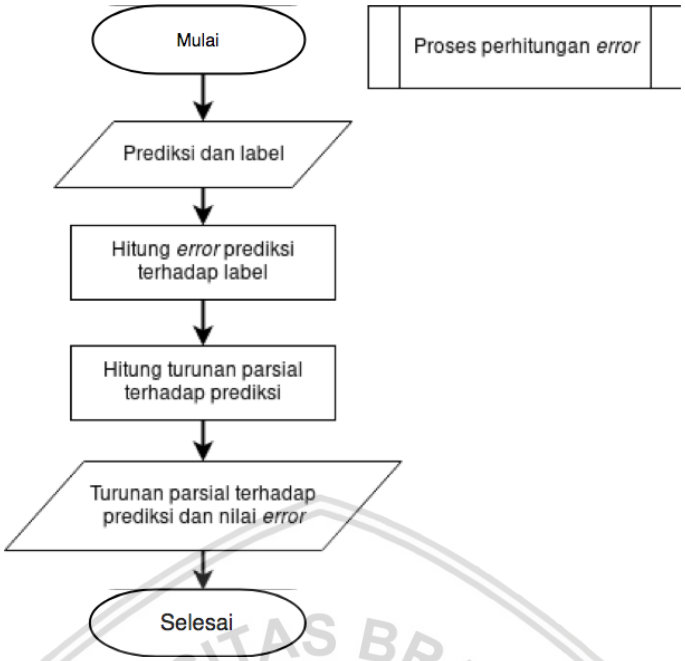


Gambar 4.6 Proses propagasi maju

4.4.5 Proses Propagasi Mundur

Proses propagasi mundur menghitung gradien dengan mencari turunan parsial terhadap seluruh bobot di setiap lapisan. Diagram alir dari algoritme dapat dilihat di Gambar 4.8.

Berdasarkan diagram alir pada Gambar 4.8, Proses ini mendapat *input* dari fungsi perhitungan *error*. Dalam setiap lapisan selain menghitung turunan parsial bobot, juga menghitung turunan parsial *input*. Turunan parsial *input* berguna untuk menghitung turunan parsial bobot pada lapisan sebelumnya. Proses ini dilakukan berulang-ulang hingga pada lapisan pertama. Setelah lapisan pertama, gradien yang telah dihitung akan digunakan untuk melakukan modifikasi bobot menggunakan teknik *gradient descent*.



Gambar 4.7 Proses perhitungan *error*

4.4.6 Lapisan Convolution

Lapisan *convolution* memiliki 3 parameter, yaitu *filter, padding* dan *stride* atau jumlah pergeseran *filter*. Diagram alir dalam mode propagasi maju dapat dilihat pada Gambar 4.9 dan dalam mode propagasi mundur pada Gambar 4.10 dan 4.11.

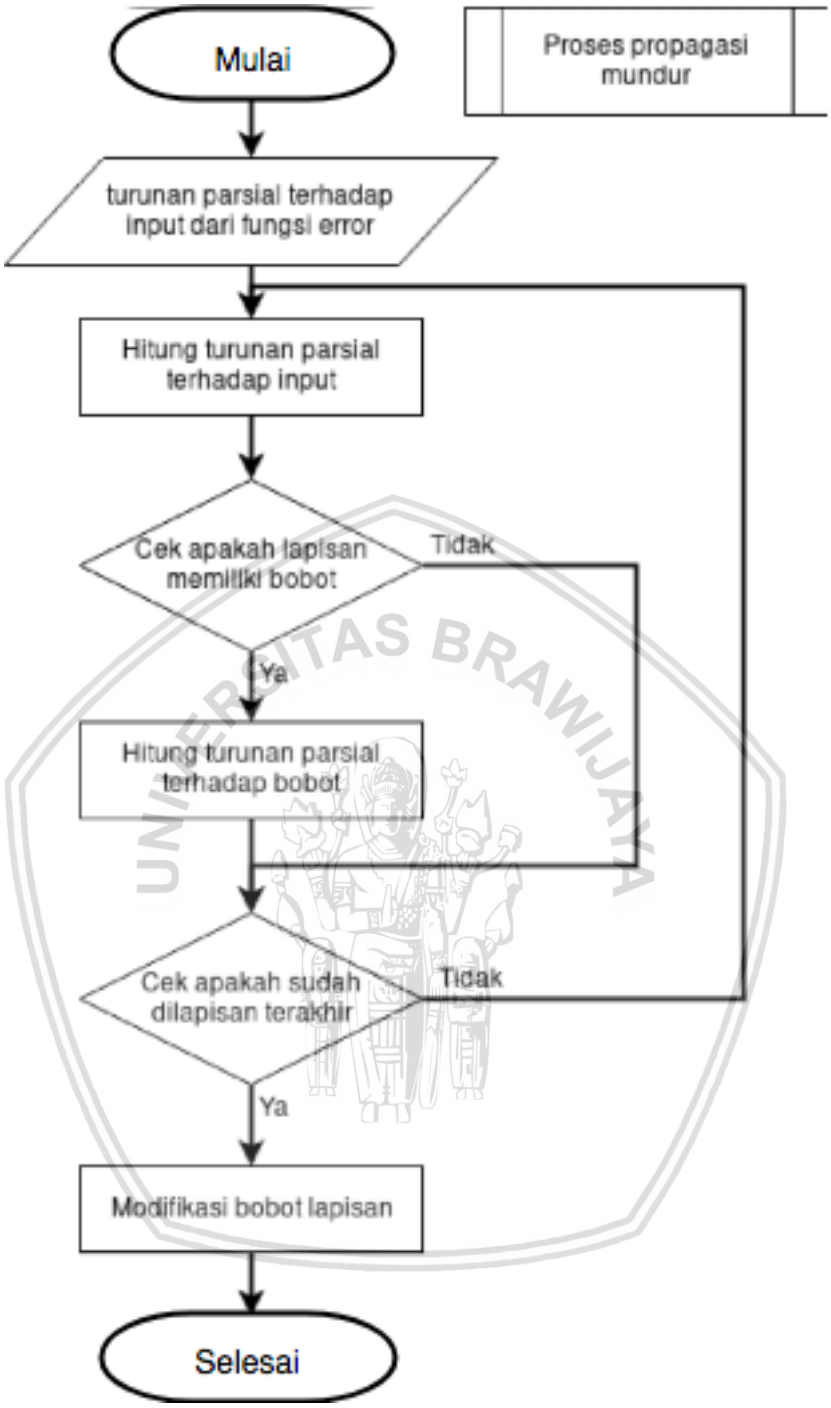
Berdasarkan diagram alir pada Gambar 4.9, algoritme melakukan perulangan secara horizontal dan vertikal pada matriks input untuk menerapkan masing-masing filter. Dari setiap bagian yang dilewati filter, menyimpan hasil *convolution* dari sampel *input* dan *filter* pada indeks *i* dan *j*. Kemudian menjumlahkan *output* convolution dengan *bias*.

Berdasarkan diagram alir pada Gambar 4.10 dan 4.11, algoritme melakukan perulangan secara horizontal dan vertikal pada matriks input untuk menerapkan masing-masing *filter*. Pada setiap sampel *input*, masing-masing dikalikan dengan turunan parsial lapisan sebelumnya untuk menghitung turunan parsial terhadap bobot lapisan.

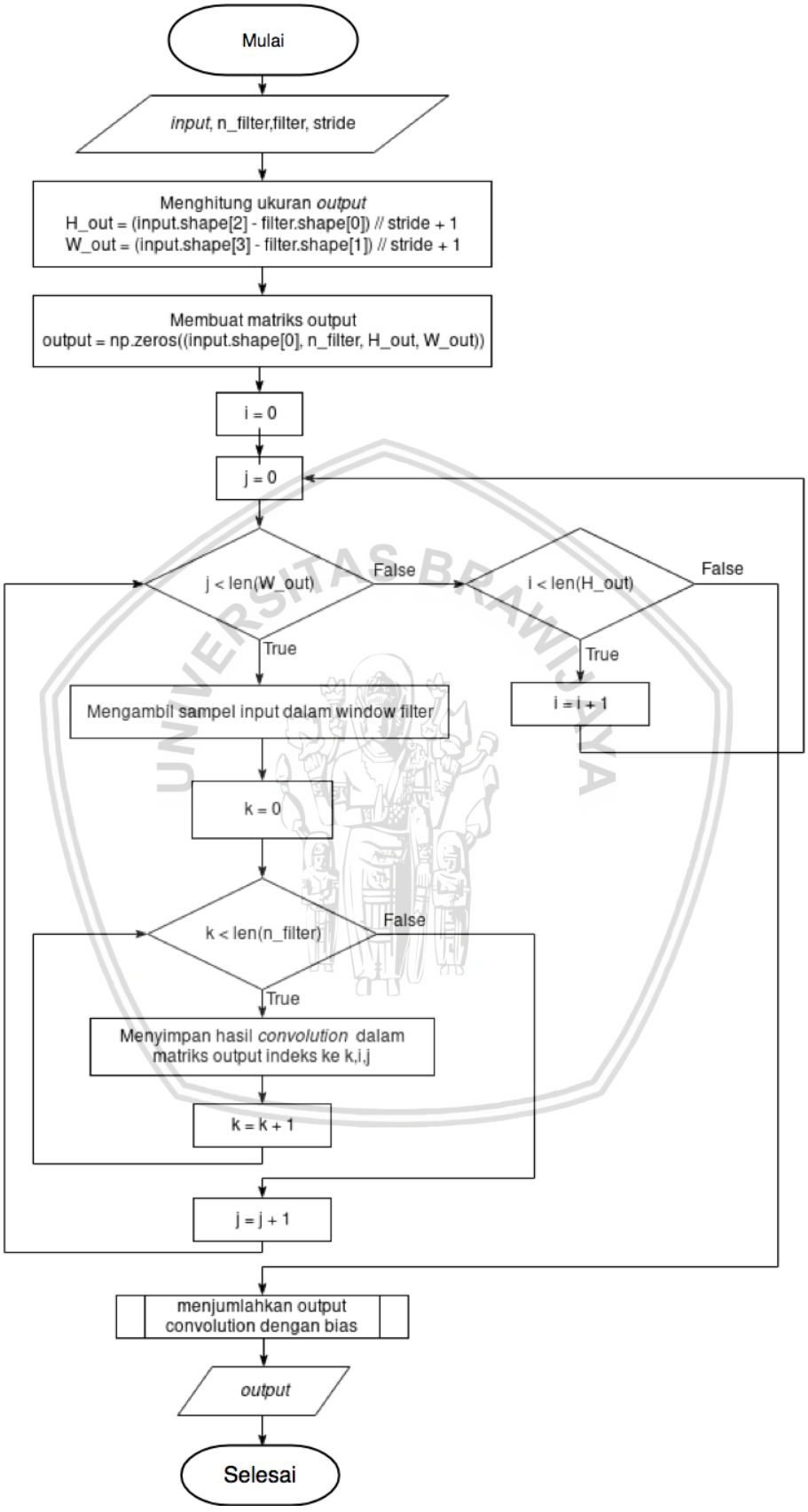
4.4.7 Lapisan ReLU

Lapisan ReLU mengubah nilai negatif menjadi 0 pada *input*. Pada diagram alir dalam Gambar 4.12, lapisan menerima *input* berupa matriks dan mode. Dalam propagasi maju, karena ReLU menggunakan fungsi *max* maka fungsi ini tidak dapat diturunkan. Sehingga itu fungsi *max* diganti dengan matriks biner *mask*. Dengan menggunakan perkalian matriks, maka turunan parsial terhadap inputnya dapat dihitung.

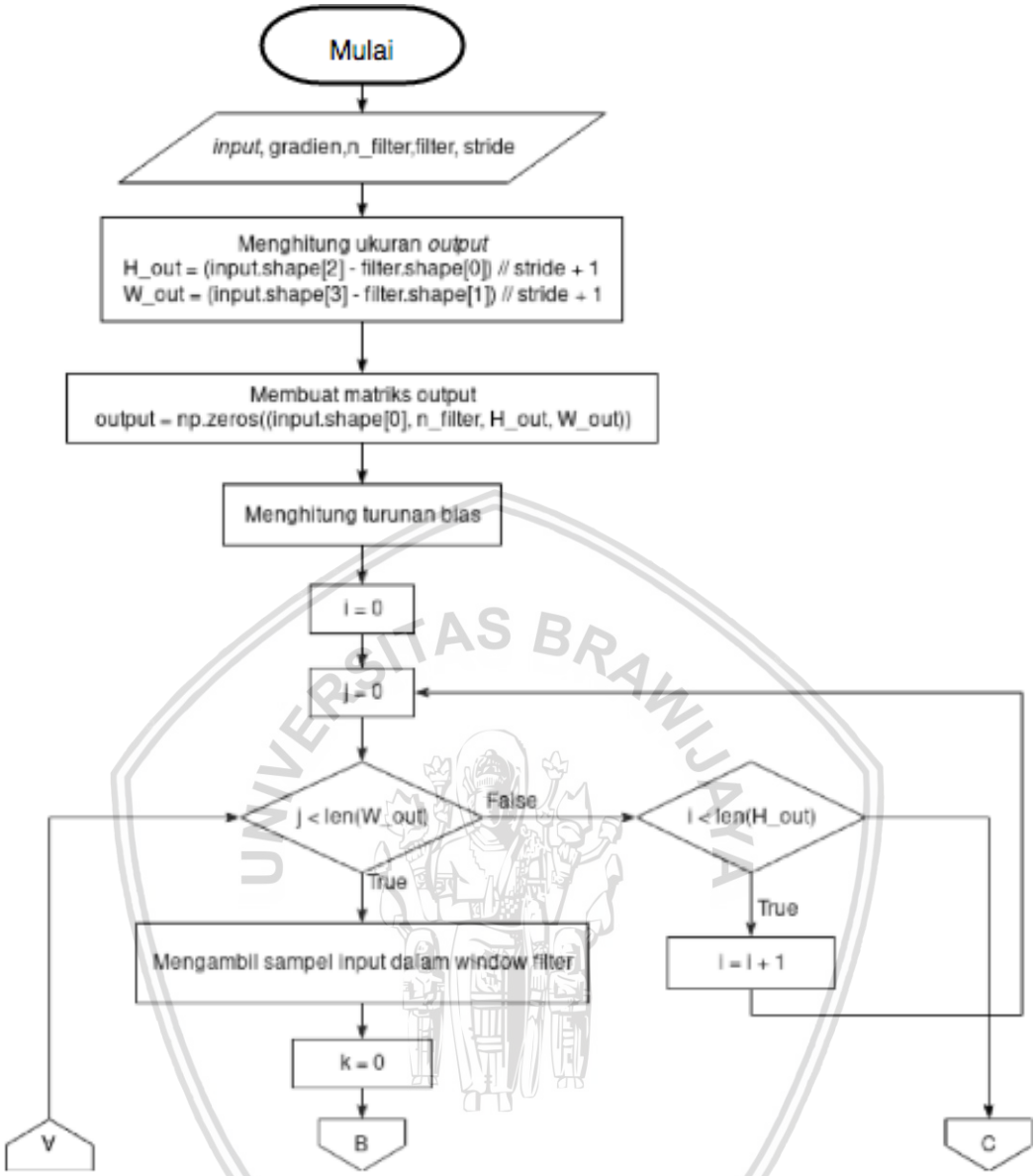
Dalam propagasi mundur, ReLU mengambil matriks *mask* saat mode propagasi maju untuk menghitung turunan parsial terhadap *input*.



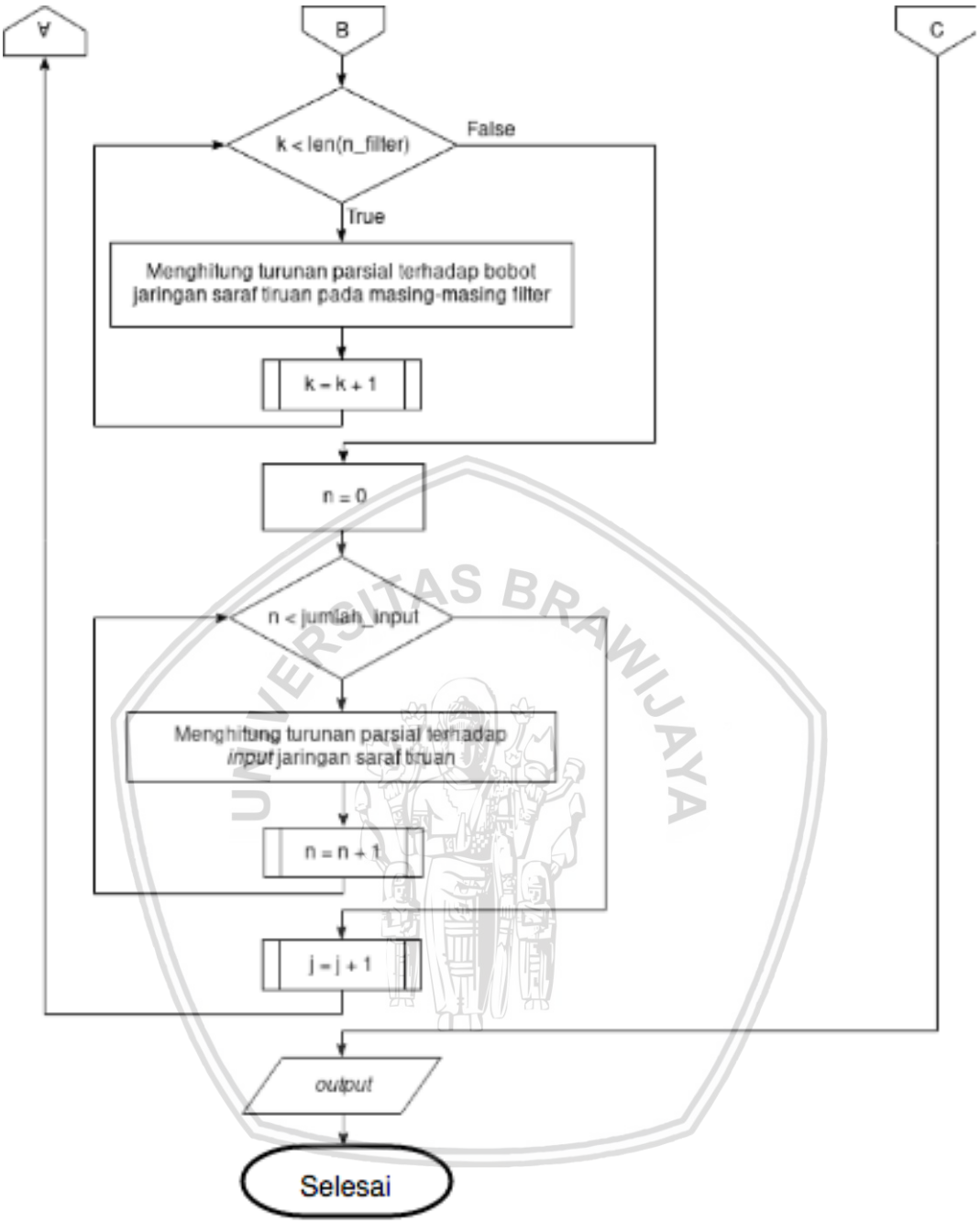
Gambar 4.8 Proses propagasi mundur



Gambar 4.9 Mode propagasi maju dalam lapisan convolution



Gambar 4.10 Mode propagasi mundur dalam lapisan convolution bagian 1



Gambar 4.11 Mode propagasi mundur dalam lapisan convolution bagian 2

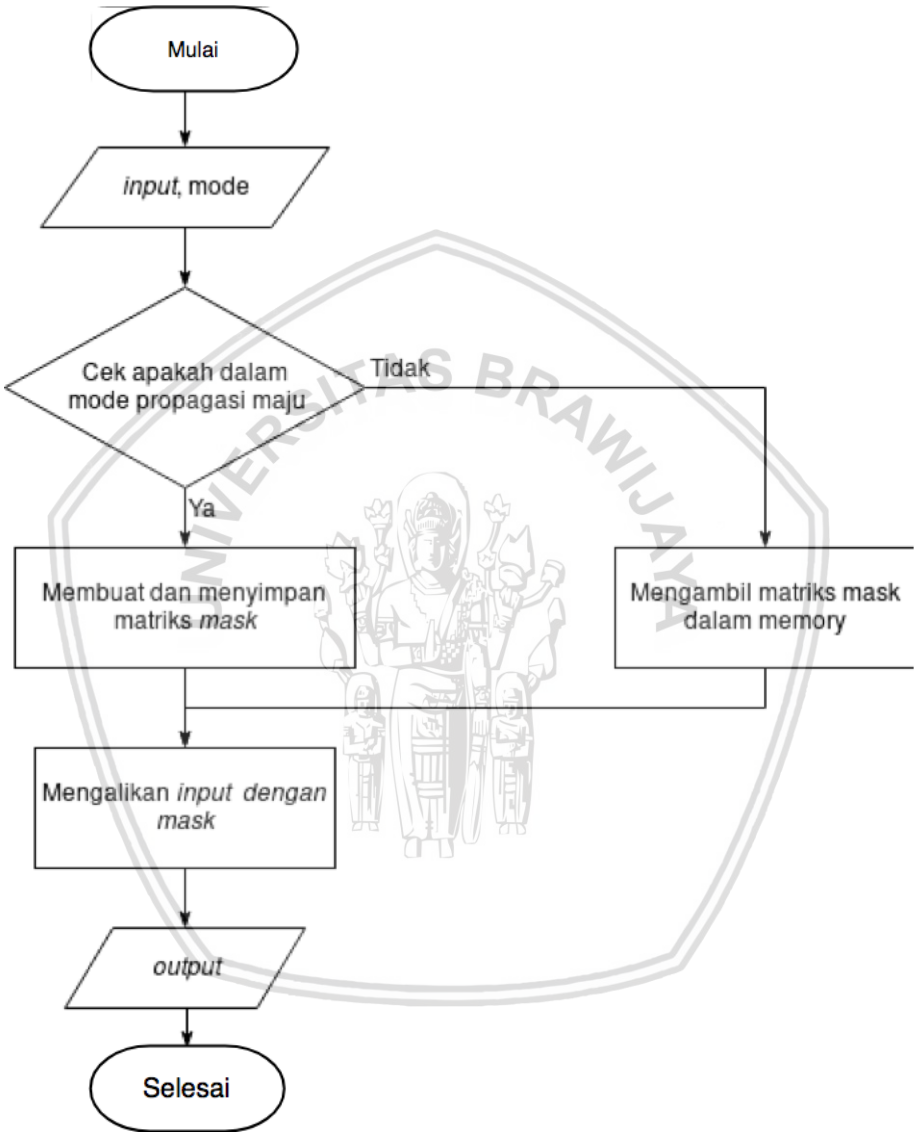
4.4.8 Lapisan Max Pool

Lapisan *max pool* melakukan *subsampling* pada *activation map*. *Max pool* memiliki parameter *filter* dan *stride* atau jumlah pergeseran *filter*. Diagram alir dalam mode propagasi maju dapat dilihat pada Gambar 4.13 dan dalam mode propagasi mundur pada Gambar 4.14.

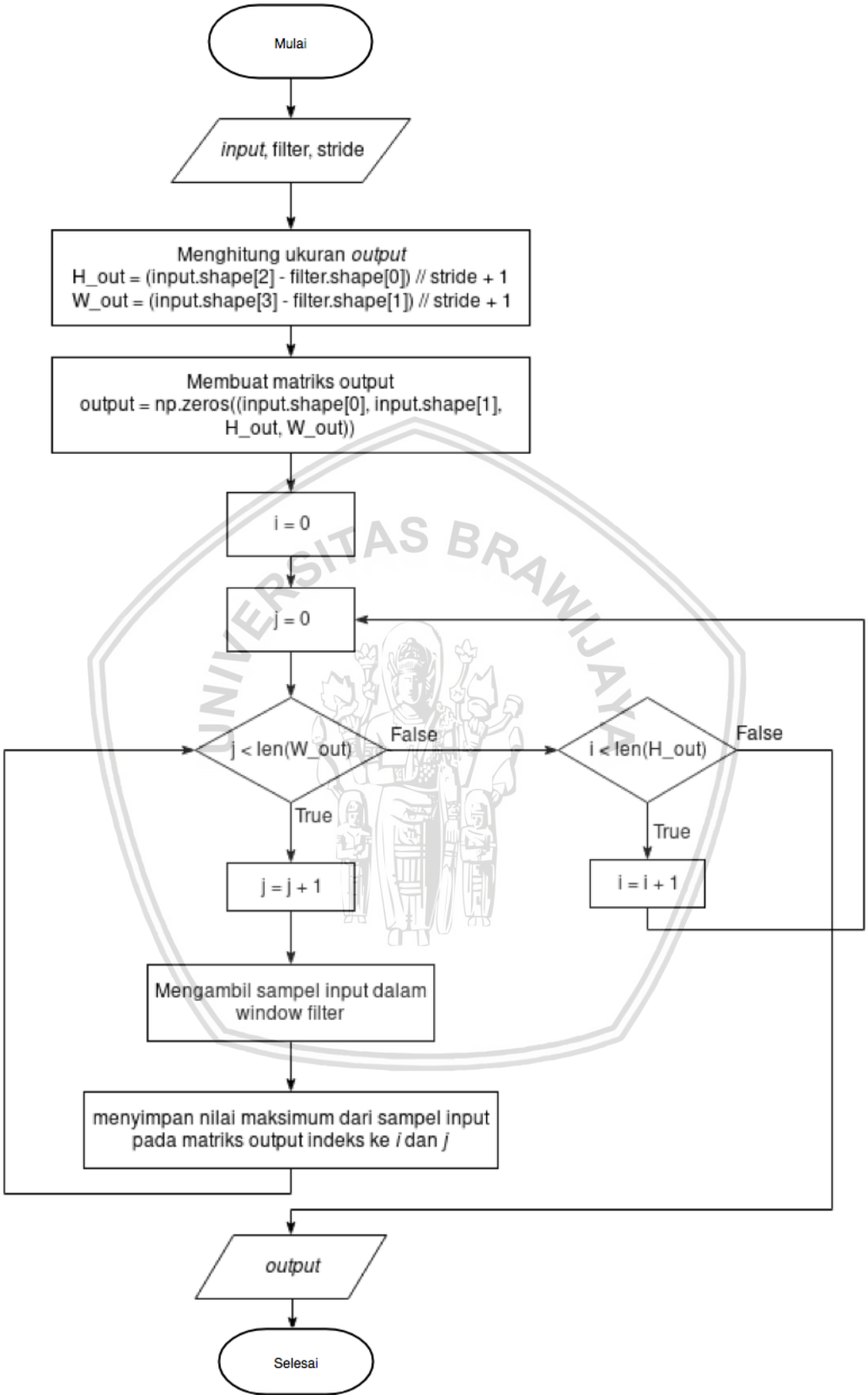
Berdasarkan diagram alir pada Gambar 4.13, algoritme melakukan perulangan secara horizontal dan vertikal pada matriks input untuk menerapkan

masing-masing filter. Dari setiap bagian yang dilewati filter, diambil nilai tertinggi dari sampel *input* untuk disimpan dalam matriks *output*.

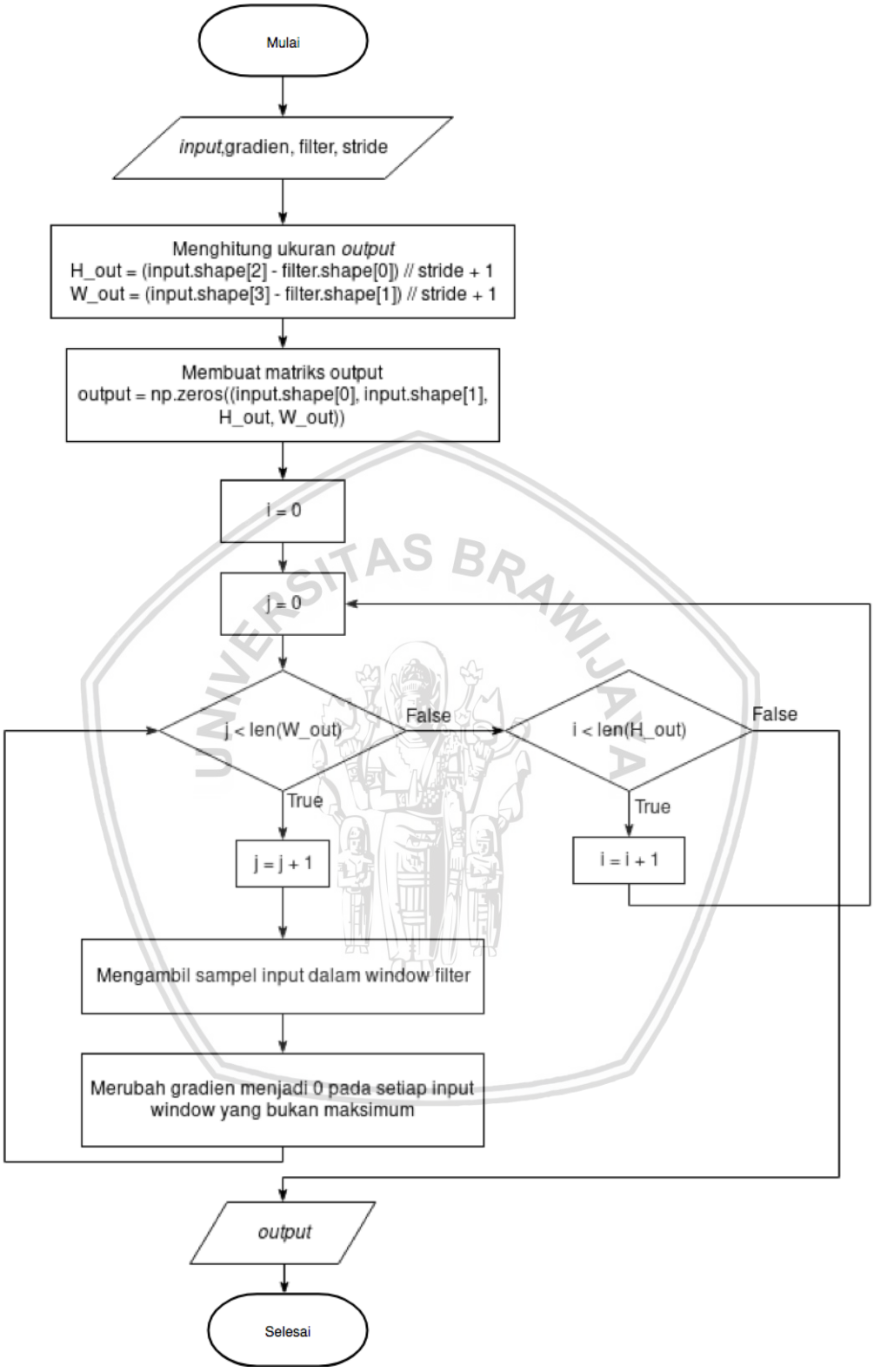
Berdasarkan diagram alir pada Gambar 4.14, algoritme melakukan perulangan secara horizontal dan vertikal pada matriks input untuk menerapkan masing-masing filter. Pada setiap filter, turunan parsial diteruskan untuk setiap elemen yang memiliki nilai tertinggi pada masing-masing sampel *input*.



Gambar 4.12 Lapisan ReLU



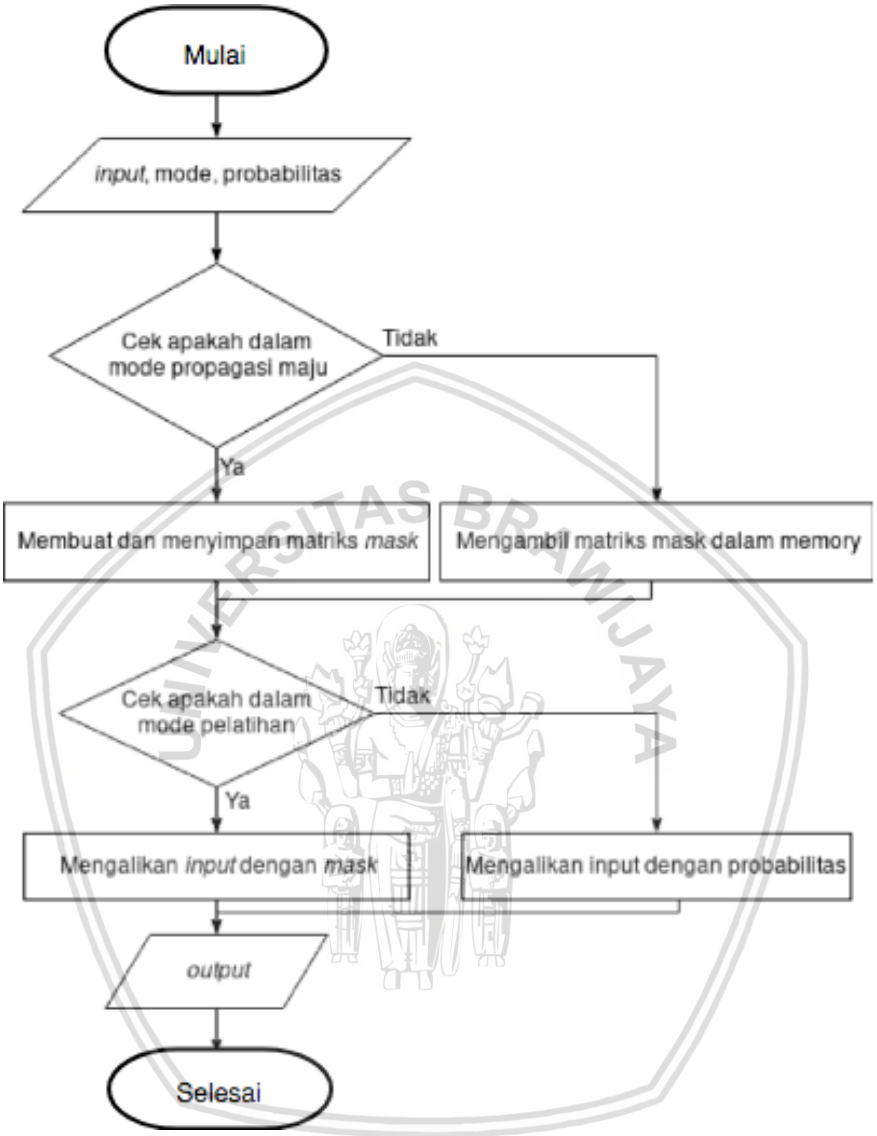
Gambar 4.13 Lapisan max pool saat propagasi maju



Gambar 4.14 Lapisan max pool saat propagasi mundur

4.4.9 Lapisan *Dropout*

Lapisan *dropout* mengubah beberapa neuron menjadi 0. Neuron yang dimatikan dipilih secara acak dengan parameter probabilitas. Diagram alir dapat dilihat di Gambar 4.15.



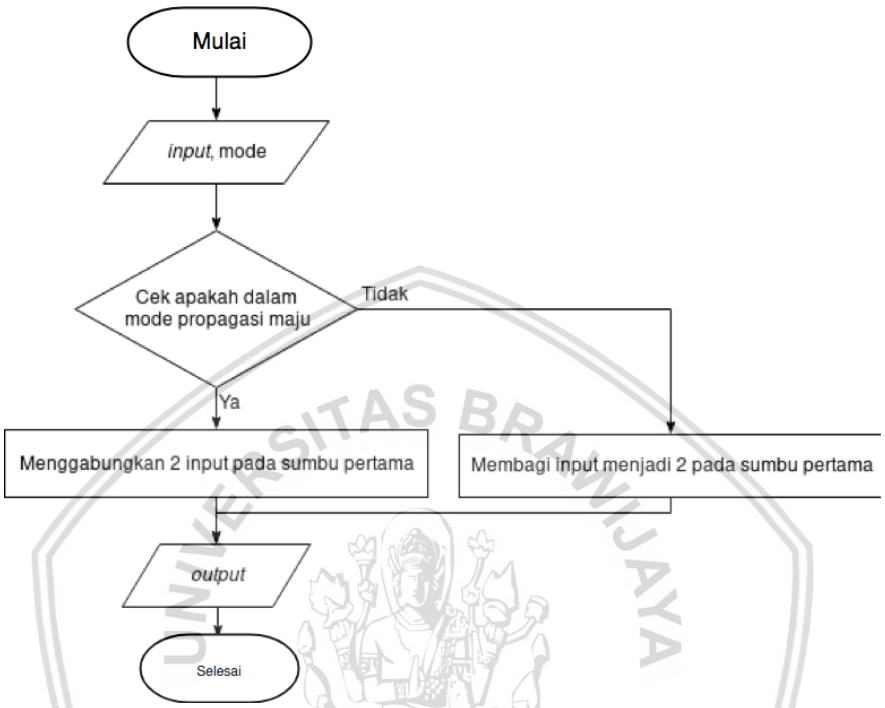
Gambar 4.15 Lapisan *Dropout*

Diagram alir dalam Gambar 4.15, lapisan menerima *input* berupa matriks, mode dan probabilitas. Dalam propagasi maju, matriks biner *mask* dengan probabilitas yang sudah ditentukan digunakan untuk menentukan elemen matriks input yang akan dirubah 0. Kemudian saat mode pelatihan, matriks input dikalikan dengan matriks *mask* untuk mendapatkan *output* dari lapisan. Ketika mode prediksi, input cukup dikalikan dengan probabilitas agar memiliki ukuran matriks yang sama ketika pelatihan.

Dalam propagasi mundur, *dropout* mengambil matriks *mask* saat mode propagasi maju untuk menghitung turunan parsial terhadap *input*. Kemudian melakukan proses yang sama seperti tahap propagasi maju untuk menentukan *output* dari lapisan.

4.4.10 Lapisan *Merge*

Lapisan *merge* menggabungkan matriks dengan ukuran yang sama pada sumbu pertama (*activation map*). Fungsi ini tidak dapat diturunkan, sehingga dalam propagasi mundur gradien dipisah pada sumbu pertama dan dibagikan ke masing-masing *input*. Diagram alir dapat dilihat pada Gambar 4.16.



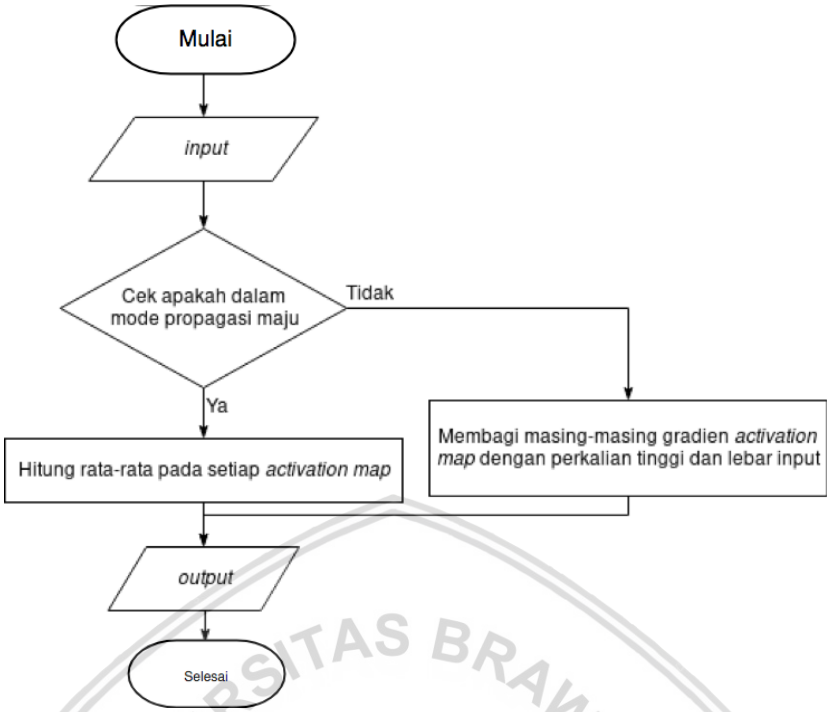
Gambar 4.16 Lapisan *merge*

Berdasarkan diagram alir pada Gambar 4.16, lapisan *merge* memiliki 2 *input* saat mode propagasi maju dan menggabungkan keduanya pada sumbu matriks pertama. Ketika mode propagasi mundur, lapisan mendapatkan 1 *input* kemudian membagi menjadi 2 pada sumbu pertama.

4.4.11 Lapisan *Global Average Pool*

Lapisan *global average pool* hampir sama dengan *max pool* kecuali filter berukuran sama dengan matriks *input* dan seluruh nilai *input* dihitung rata-ratanya. Diagram alir dapat dilihat pada Gambar 4.17.

Berdasarkan diagram alir pada Gambar 4.17, dalam mode propagasi maju lapisan ini merata-rata sumbu ke 2&3 (tinggi dan lebar) masing-masing *activation map* dari matriks *input*. Kemudian dalam mode propagasi mundur turunan parsian dari lapisan setelahnya dibagi dengan hasil perkalian ukuran tinggi dan lebar dari *input*. Sehingga turunan parsial yang diteruskan ke lapisan selanjutnya memiliki nilai yang sama dalam setiap *activation map*.



Gambar 4.17 Lapisan *global average pool*

4.1 Perhitungan Manual

Sub bab ini menjelaskan perhitungan manual untuk memberikan gambaran umum berkaitan dengan alur proses perhitungan yang akan dikembangkan pada algoritme.

Untuk langkah-langkah perhitungan manual sistem adalah sebagai berikut:

Langkah 1. Menentukan *dataset* dan model yang dipakai

Pada proses manualisasi, *dataset* dan model yang akan digunakan dengan ketentuan:

a) Jumlah data hanya 1 gambar dari data uji

b) Sampel merupakan gambar dengan ukuran yang lebih kecil yaitu 5x5

c) Berlabel belok kiri

Data yang akan digunakan dalam proses manualisasi ini dapat dilihat pada Gambar 4.18 dan Tabel 4.2.

Gambar 4.18 Input jaringan saraf tiruan

Tabel 4.2 Contoh data

Channel	Nilai
1 (*Red*)	$\begin{bmatrix} 106 & 79 & 126 & 107 & 112 \\ 80 & 80 & 104 & 81 & 118 \\ 78 & 77 & 81 & 79 & 77 \\ 76 & 79 & 81 & 73 & 69 \\ 76 & 82 & 79 & 71 & 80 \end{bmatrix}$
2 (*Green*)	$\begin{bmatrix} 109 & 80 & 125 & 106 & 108 \\ 88 & 83 & 109 & 84 & 119 \\ 81 & 78 & 83 & 81 & 78 \\ 79 & 80 & 85 & 76 & 70 \\ 79 & 86 & 83 & 73 & 88 \end{bmatrix}$
3 (*Blue*)	$\begin{bmatrix} 109 & 86 & 128 & 89 & 98 \\ 88 & 88 & 111 & 88 & 118 \\ 87 & 84 & 88 & 87 & 83 \\ 83 & 86 & 90 & 79 & 75 \\ 82 & 92 & 88 & 79 & 98 \end{bmatrix}$

Langkah 2 Inisialisasi bobot lapisan pertama secara acak

Tabel 4.3 Inisialisasi bobot lapisan *convolution* 1

No. Filter	Channel	Bobot Filter	Bias
1	1	$\begin{bmatrix} -0{,}197 & -0{,}192 \\ 0{,}0463 & -0{,}063 \end{bmatrix}$	0
	2	$\begin{bmatrix} 0{,}1726 & -0{,}030 \\ 0{,}0366 & 0{,}0192 \end{bmatrix}$	
	3	$\begin{bmatrix} -0{,}125 & -0{,}174 \\ 0{,}0564 & -0{,}045 \end{bmatrix}$	
2	1	$\begin{bmatrix} 0{,}0253 & -0{,}099 \\ 0{,}0888 & -0{,}091 \end{bmatrix}$	0
	2	$\begin{bmatrix} -0{,}175 & 0{,}0645 \\ -0{,}018 & -0{,}015 \end{bmatrix}$	
	3	$\begin{bmatrix} 0{,}0825 & -0{,}034 \\ 0{,}0214 & 0{,}0196 \end{bmatrix}$	
3	1	$\begin{bmatrix} 0{,}0308 & 0{,}0869 \\ -0{,}026 & 0{,}1560 \end{bmatrix}$	0

	2	$\begin{bmatrix}-0{,}107 & -0{,}218\\ -0{,}104 & 0{,}0112\end{bmatrix}$	
	3	$\begin{bmatrix}-0{,}090 & -0{,}458\\ -0{,}076 & 0{,}1598\end{bmatrix}$	
4	1	$\begin{bmatrix}-0{,}036 & -0{,}014\\ -0{,}246 & -0{,}026\end{bmatrix}$	0
	2	$\begin{bmatrix}0{,}1372 & 0{,}0551\\ -0{,}218 & 0{,}0610\end{bmatrix}$	
	3	$\begin{bmatrix}0{,}1812 & -0{,}162\\ 0{,}1185 & 0{,}1147\end{bmatrix}$	

Langkah 3 Hitung *activation map* lapisan *convolution* 1

Berdasarkan *input* dan bobot di atas, operasi *convolution* menghasilkan *activation map* sebagai berikut.

$$
\begin{aligned}
O_{111} &= \sum_{c=1}^{3} grandsum(I_{c(1,1)(2,2)} w_{1c}) + b_c \\
&= grandsum(I_{1(1,1)(2,2)} w_{11}) + 0 + grandsum(I_{2(1,1)(2,2)} w_{12}) + 0 \\
&\quad + grandsum(I_{3(1,1)(2,2)} w_{13}) + 0 \\
&= grandsum\left(\begin{bmatrix}106 & 79\\ 80 & 80\end{bmatrix}\begin{bmatrix}-0{,}197 & -0{,}192\\ 0{,}0463 & -0{,}063\end{bmatrix}\right) + 0 \\
&\quad + grandsum\left(\begin{bmatrix}109 & 80\\ 88 & 83\end{bmatrix}\begin{bmatrix}0{,}1726 & -0{,}030\\ 0{,}0366 & 0{,}0192\end{bmatrix}\right) + 0 \\
&\quad + grandsum\left(\begin{bmatrix}109 & 86\\ 88 & 88\end{bmatrix}\begin{bmatrix}-0{,}125 & -0{,}174\\ 0{,}0564 & -0{,}045\end{bmatrix}\right) + 0 \\
&= grandsum\left(\begin{bmatrix}-20{,}983 & -15.210\\ 3.7090 & -5.054\end{bmatrix}\right) \\
&\quad + grandsum\left(\begin{bmatrix}18.8175 & -2.408\\ 3.0441 & 1.5956\end{bmatrix}\right) \\
&\quad + grandsum\left(\begin{bmatrix}-13.701 & -14.968\\ 4.9685 & -4.001\end{bmatrix}\right) \\
&= -37.5402 + 21.0488 + (-27.7032) \\
&= -44.1938 \\
O_{112} &= \sum_{c=1}^{3} grandsum(I_{c(2,1)(2,3)} w_{1c}) + b_c \\
&= grandsum(I_{1(2,1)(2,3)} w_{11}) + 0 + grandsum(I_{2(2,1)(2,3)} w_{12}) + 0 \\
&\quad + grandsum(I_{3(2,1)(2,3)} w_{13}) + 0 \\
&= grandsum\left(\begin{bmatrix}79 & 126\\ 80 & 104\end{bmatrix}\begin{bmatrix}-0{,}197 & -0{,}192\\ 0{,}0463 & -0{,}063\end{bmatrix}\right) + 0 \\
&\quad + grandsum\left(\begin{bmatrix}80 & 125\\ 83 & 109\end{bmatrix}\begin{bmatrix}0{,}1726 & -0{,}030\\ 0{,}0366 & 0{,}0192\end{bmatrix}\right) + 0 \\
&\quad + grandsum\left(\begin{bmatrix}86 & 128\\ 88 & 111\end{bmatrix}\begin{bmatrix}-0{,}125 & -0{,}174\\ 0{,}0564 & -0{,}045\end{bmatrix}\right) + 0
\end{aligned}
$$

$$= grandsum\left(\begin{bmatrix} -15.638 & -24.260 \\ 3.7090 & -6.571 \end{bmatrix}\right) + grandsum\left(\begin{bmatrix} 13.8110 & -3.763 \\ 3.0441 & 2.0955 \end{bmatrix}\right) + grandsum\left(\begin{bmatrix} -10{,}810 & -22.278 \\ 4.9685 & -5.047 \end{bmatrix}\right)$$
$$= -42.7611 + 15.1874 + (-33.1673)$$
$$= -60{,}741$$

...

$$\boldsymbol{O_{444} = \sum_{c=1}^{3} grandsum(I_{c(4,4)(5,5)} w_{4c}) + b_c}$$
$$= \boldsymbol{grandsum(I_{1(4,4)(5,5)} w_{11}) + 0 + grandsum(I_{2(4,4)(5,5)} w_{12}) + 0 + grandsum(I_{3(4,4)(5,5)} w_{13}) + 0}$$
$$= \boldsymbol{grandsum\left(\begin{bmatrix} 79 & 126 \\ 80 & 104 \end{bmatrix}\begin{bmatrix} -0,036 & -0,014 \\ -0,246 & -0,026 \end{bmatrix}\right) + 0 + grandsum\left(\begin{bmatrix} 80 & 125 \\ 83 & 109 \end{bmatrix}\begin{bmatrix} 0,1372 & 0,0551 \\ -0,218 & 0,0610 \end{bmatrix}\right) + 0 + grandsum\left(\begin{bmatrix} 86 & 128 \\ 88 & 111 \end{bmatrix}\begin{bmatrix} 0,1812 & -0,162 \\ 0,1185 & 0,1147 \end{bmatrix}\right) + 0}$$
$$= \boldsymbol{-42.7611 + 15.1874 + (-33.1673)}$$
$$= \boldsymbol{3.147}$$

Berdasarkan perhitungan di atas dihasilkan *activation map* seperti pada Tabel 4.4.

Tabel 4.4 Activation map lapisan *convolution* 1

Activation Map	Nilai
1	$\begin{bmatrix} -44.193 & -60{,}741 & -51.154 & -54.671 \\ -41.473 & -51.545 & -44.786 & -55.029 \\ -40{,}444 & -41.870 & -40{,}348 & -40{,}003 \\ -41.087 & -41.704 & -37.766 & -37.096 \end{bmatrix}$
2	$\begin{bmatrix} -12.463 & -15.265 & -12.366 & -18.788 \\ -10{,}148 & -12.091 & -12.165 & -12.694 \\ -10{,}218 & -10{,}042 & -9.566 & -9.607 \\ -10{,}654 & -9.660 & -9.163 & -9.940 \end{bmatrix}$
3	$\begin{bmatrix} -58.480 & -71.441 & -70{,}887 & -54.085 \\ -56.900 & -69.172 & -60{,}709 & -75.509 \\ -52.764 & -54.190 & -58.232 & -55.006 \\ -51.859 & -57.700 & -54.314 & -43.240 \end{bmatrix}$
4	$\begin{bmatrix} 5.7210 & -2.865 & 2.3152 & 4.9405 \\ -0{,}742 & -2.193 & 5.5658 & -4.453 \\ 0{,}0494 & -1.193 & -2.562 & -0{,}516 \\ -0{,}262 & -2.483 & -0{,}251 & 3.1470 \end{bmatrix}$

Langkah 4 Hitung hasil lapisan *ReLU*

Lapisan ini dihitung hanya dengan mengganti nilai negatif menjadi 0 seperti pada Tabel 4.5.

Tabel 4.5 *Activation map* lapisan *ReLU* 1

Activation Map	Nilai
1	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
3	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
4	$\begin{bmatrix} 5.72100 & 0 & 2.3152 & 4.9405 \\ 0 & 0 & 5.5658 & 0 \\ 0{,}0494 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3.1470 \end{bmatrix}$

Langkah 5 Hitung hasil lapisan *pooling*

Menghitung lapisan *pooling* terdiri dari 2 tahap yaitu menghitung ukuran *output* dan nilai *output*.

1. Hitung tinggi (t) dan lebar (l) matriks *output* (O)

$$t = \frac{(tinggi\ input - tinggi\ filter)}{stride} + 1$$
$$= \frac{4-2}{2} + 1$$
$$= 2$$
$$l = t = 2$$

2. Menghitung nilai *output* (O)

$$O_{111} = \max\left(I_{1(1,1)(2,2)}\right)$$
$$= \max\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$
$$= 0$$

...

$$O_{212} = \max\left(I_{2(1,3)(2,4)}\right)$$
$$= \max\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$
$$= 0$$

...

$$O_{422} = \max\left(I_{4(3,3)(4,4)}\right)$$
$$= \max\begin{pmatrix} 0 & 0 \\ 0 & 3.1470 \end{pmatrix}$$
$$= 3.1470$$

Berdasarkan perhitungan di atas, hasil dari lapisan *pooling* adalah pada Tabel 4.5.

Tabel 4.6 Hasil lapisan pooling 1

Activation Map	Nilai
1	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
3	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
4	$\begin{bmatrix} 5.72100 & 5.5658 \\ 0{,}0494 & 3.1470 \end{bmatrix}$

Langkah 6 Inisialisasi bobot dan hitung *activation map* lapisan *convolution* 2

Langkah ini terdiri dari 2 tahap yaitu inisialisasi dan perhitungan hasil. Saat inisialisasi bobot filter diisi secara acak seperti pada Tabel 4.6. Kemudian melakukan perhitungan hasil *activation map*.

Tabel 4.7 Inisialisasi bobot lapisan *convolution* 2

No. Filter	Channel	Bobot Filter	Bias
1	1	0,0016	0
	2	0,1415	
	3	0,0148	
	4	0,1155	
2	1	-0,0452	0
	2	-0,0041	
	3	0,0924	
	4	0,0630	
3	1	-0,0251	0
	2	0,0221	
	3	0,0148	
	4	0,1417	
4	1	0,0608	0

	2	-0,0032	
	3	0,0989	
	4	0,0525	

$$O_{111} = \sum_{c=1}^{4} I_{c11} W_{1c} + b_c$$
$$= (0\ x\ 0{,}0016 + 0) + (0\ x\ 0{,}1415 + 0) + (0\ x\ 0{,}0148 + 0) + (5.7210\ x\ 0{,}1155 + 0)$$
$$= 0{,}6608$$

$$O_{112} = \sum_{c=1}^{4} I_{c12} W_{1c} + b_c$$
$$= (0\ x\ 0{,}0016 + 0) + (0\ x\ 0{,}1415 + 0) + (0\ x\ 0{,}0148 + 0) + (5.5658\ x\ 0{,}1155 + 0)$$
$$= 0{,}6428$$

...

$$O_{221} = \sum_{c=1}^{4} I_{c21} W_{2c} + b_c$$
$$= (0\ x\ -0{,}0452 + 0) + (0\ x\ -0{,}0041 + 0) + (0\ x\ 0{,}0924 + 0) + (0{,}0494\ x\ 0{,}0630 + 0)$$
$$= 0{,}0031$$

...

$$O_{422} = \sum_{c=1}^{4} I_{c22} W_{4c} + b_c$$
$$= (0\ x\ 0{,}0608 + 0) + (0\ x\ -0{,}0032 + 0) + (0\ x\ 0{,}0989 + 0) + (3.1470\ x\ 0{,}0525 + 0)$$
$$= 0{,}1653$$

Berdasarkan perhitungan di atas, *activation map* dari lapisan convolution 2 adalah pada Tabel 4.7.

Tabel 4.8 *Activation map* lapisan *convolution* 2

Activation Map	Nilai
1	$\begin{bmatrix} 0{,}6608 & 0{,}6428 \\ 0{,}0057 & 0{,}3635 \end{bmatrix}$
2	$\begin{bmatrix} 0{,}3608 & 0{,}3510 \\ 0{,}0031 & 0{,}1985 \end{bmatrix}$
3	$\begin{bmatrix} 0{,}8112 & 0{,}7892 \\ 0{,}0070 & 0{,}4462 \end{bmatrix}$

4	$\begin{bmatrix} 0{,}3005 & 0{,}2924 \\ 0{,}0025 & 0{,}1653 \end{bmatrix}$

Langkah 7 Hitung prediksi pada lapisan *global average pool*

$$O_1 = \frac{\sum_{i=1}^{4} I_{1i}}{4}$$
$$= \frac{1}{4} \text{ x } (0{,}6608 + 0{,}6428 + 0{,}0057 + 0{,}3635)$$
$$= 0{,}4182$$
$$O_2 = \frac{\sum_{i=1}^{4} I_{2i}}{4}$$
$$= \frac{1}{4} \text{ x } (0{,}3608 + 0{,}3510 + 0{,}0031 + 0{,}1985)$$
$$= 0{,}2284$$
$$O_3 = \frac{\sum_{i=1}^{4} I_{3i}}{4}$$
$$= \frac{1}{4} \text{ x } (0{,}8112 + 0{,}7892 + 0{,}0070 + 0{,}4462)$$
$$= 0{,}5134$$
$$O_4 = \frac{\sum_{i=1}^{4} I_{4i}}{4}$$
$$= \frac{1}{4} \text{ x } (0{,}3005 + 0{,}2924 + 0{,}0025 + 0{,}1653)$$
$$= 0{,}1902$$

Berdasarkan hasil perhitungan di atas, hasil lapisan *global average pool* yang merupakan hasil akhir prediksi adalah pada Tabel 4.8.

Tabel 4.9 Hasil lapisan *global average pool*

Label	Nilai
1	0,4182
2	0,2284
3	0,5134
4	0,1902

Langkah 8 Menghitung nilai *error*

Nilai error dihitung menggunakan metode *cross entropy* menggunakan data pada Tabel 4.10.

Tabel 4.10 Nilai prediksi dan label

Nama Label	Nilai Prediksi	*Ground Truth*
Kiri	0,4182	1
Kanan	0,2284	0
Booster	0,5134	0
Lurus	0,1902	0

$$
\begin{aligned}
xe = & -(\ln(0{,}4182) * 1) + (\ln(0{,}2284) * 0) + (\ln(0{,}5134) * 0) \\
& + (\ln(0{,}1902) * 0) \\
= & -(\ln(0{,}4182) * 1) \\
= & -(-0{,}8717) \\
= & \ 0{,}8717
\end{aligned}
$$

Berdasarkan perhitungan *cross-entropy* di atas, *xe* adalah nilai *cross-entropy* yang akan digunakan sebagai nilai *error*. Sehingga dalam contoh kasus ini nilai *error* jaringan saraf tiruan adalah 0,8717.

BAB 5 IMPLEMENTASI

Proses implementasi pada bab ini adalah tahap pembuatan sistem "Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional di Dalam *Game SuperTuxKart*" berdasarkan perancangan yang telah dibuat sebelumnya. Dalam bab ini akan berisi mengenai perangkat keras dan perangkat lunak yang digunakan pada proses implementasi sistem, batasan implementasi, implementasi algoritme yang digunakan pada sistem, dan terakhir implementasi antarmuka sistem yang digunakan.

5.1 Perangkat Keras

Dalam mengimplementasikan sistem yang dibuat pada penelitian ini adalah dengan menggunakan perangkat keras laptop ASUS A43SJ dengan spesifikasi sebagai berikut:

1. Processor Intel(R) Core(TM) i3 CPU @2.40GHz (4 CPUs), ~2.4GHz.
2. Harddrive 500 GB.
3. Memori RAM 8192 MB.

5.2 Perangkat Lunak

Sedangkan untuk perangkat lunak yang digunakan untuk pengimplementasian sistem adalah:

1. Sistem Arch Linux 64-bit.
2. Editor Sublime Text.
3. Python 3.6.
4. *Package* Python *mss* dan *keyboard* untuk mengambil tangkapan layar dan mengendalikan *game*.

5.3 Batasan Implementasi

Batasan yang digunakan dalam mengimplementasikan sistem adalah sebagai berikut:

1. Sistem menggunakan bahasa pemrograman Python.
2. Data yang digunakan disimpan kedalam format npy untuk dimasukkan secara bertahap kedalam jaringan saraf tiruan.
3. Metode yang digunakan dalam klon perilaku adalah jaringan saraf tiruan konvolusional dengan arstitektur arsitektur *SqueezeNet*.
4. Sumber data berasal dari rekaman permainan pemain.
5. Fitur yang digunakan adalah gambar tangkapan layar pemain.
6. Kelas data terdiri data aksi pemain dalam memainkan *game*.

5.4 Implementasi Algoritme

Pada implementasi algoritme disini menjelaskan terkait kode program dari sistem yang mengacu pada perancangan proses pada bab sebelumnya. Dimana meliputi propagasi maju, perhitungan *error* dan propagasi mundur.

5.4.1 Implementasi Proses Propagasi Maju

```
1   def _forward(self, X, y=None, mode='train'):
2       # Perform forward pass
3       out = X
4       for row in self.model:
5           if type(row) is list:
6               row_out = []
7               for i, (node, name) in enumerate(row):
8                   if type(out) is tuple:
9                       row_out.append(node.forward(out[i],
    mode))
10                  else:
11                      row_out.append(node.forward(out,
    mode))
13              out = tuple(row_out)
14          else:
15              node, name = row
16              out = node.forward(out, mode)
17
18      if y is None:
19          return out
20
21      acc = (np.argmax(out, axis=1) == y).mean()
22      loss, dout = self.loss_fn(out, y)
23
24      return loss, acc, dout
```

Source Code 5.1 Proses propagasi maju

Penjelasan Source Code 5.1, pada baris ke:

4. *Loop* dari lapisan awal jaringan saraf tiruan sampai akhir.
5. Cek lapisan selanjutnya bercabang atau tidak.
7. Loop terhadap seluruh cabang lapisan.
8. Cek lapisan selanjutnya bercabang atau tidak.
9. Jika lapisan sebelumnya bercabang, lanjutkan propagasi maju dari masing-masing cabang.
11. Jika lapisan sebelumnya tidak bercabang, masukkan *output* dari lapisan sebelumnya ke seluruh cabang.
16. Jika lapisan tidak bercabang, lanjutkan propagasi maju dari lapisan sebelumnya.
17. Jika label tidak diberikan, berikan kembalikan hasil prediksi lapisan terakhir.
21. Hitung akurasi dari prediksi.
22. Hitung *error* dan turunan parsial terhadap prediksi.

5.4.2 Implementasi Proses Perhitungan Error

```
1   def softmax_crossentropy_loss(x, y):
2       probs = np.exp(x - np.max(x, axis=1, keepdims=True))
3       probs /= np.sum(probs, axis=1, keepdims=True)
4       N = x.shape[0]
5       log = np.log(probs[np.arange(N), y])
6       loss = -np.sum(log) / N
7
8       dx = probs.copy()
9       dx[np.arange(N), y] -= 1
10      dx /= N
11      return loss, dx
```

Source Code 5.2 Proses perhitungan *error*

Penjelasan Source Code 5.2, pada baris ke:

2-3. Perhitungan fungsi *softmax*.

4-6. Perhitungan nilai *cross-entropy*.

8-10. Perhitungan turunan parsial terhadap prediksi.

5.4.3 Implementasi Proses Propagasi Mundur

```
1   def _backward(self, dout):
2       # Perform backpropagation
3       node_idx = len(self.model)
4       for row in reversed(self.model):
5           node_idx -= 1
6           if type(dout) is tuple and type(row) is list:
7               dout_row = []
8               for i, (node, name) in enumerate(row):
9                   dout_branch, grads =
    node.backward(dout[i], mode='train')
10                  dout_row.append(dout_branch)
11                  if node.trainable:
12                      self._update_param(node, name, grads)
13              dout = tuple(dout_row)
14          else:
15              if type(dout) is tuple:
16                  dout = np.add.reduce(list(dout), axis=0)
17              node, name = row
18              dout, grads = node.backward(dout,
    mode='train')
19              if node.trainable:
20                  self._update_param(node, name, grads)
```

Source Code 5.3 Implementasi proses propagasi mundur

Penjelasan Source Code 5.3, pada baris ke:

4. Loop dari lapisan terakhir jaringan saraf tiruan sampai pertama.

5. Cek apakah lapisan setelahnya dan saat ini memiliki cabang.

8. Loop terhadap seluruh cabang.

9. Melanjukan propagasi mundur ke lapisan saat ini dengan *input* turunan parsial *input* dari lapisan setelahnya.
11. Cek apakah lapisan memiliki bobot.
12. Modifikasi bobot lapisan.
15. Cek apakah lapisan setelahnya memiliki cabang.
16. Menjumlahkan seluruh gradien dari lapisan setelahnya.
18. Melanjukan propagasi mundur ke lapisan saat ini dengan *input* turunan parsial *input* dari lapisan setelahnya.
19. Cek apakah lapisan memiliki bobot.
20. Modifikasi bobot lapisan.

5.4.4 Implementasi Lapisan Convolution

```
1   class Convolution(Layer):
2       def __init__(self, output, filter_shape=(2, 2),
3   stride=1,
4                    padding='valid'):
5           super().__init__()
6           self.trainable = True
7           self.output, self.filter_shape = output,
    filter_shape
8           self.stride, self.padding = stride, padding
9           self.pad = int(filter_shape[0] - 1) // 2 if
    padding == 'same' else 0
10
11      def _get_shape(self, input_shape):
12          C, H, W = input_shape[0], input_shape[1],
    input_shape[2]
13          filter_H, filter_W = self.filter_shape[0],
    self.filter_shape[1]
14          stride, pad = self.stride, self.pad
15          self.params = {
16              'w': self.weight_scale *
    np.random.randn(self.output,
                                                    C,
    *self.filter_shape),
17              'b': self.weight_scale *
    np.zeros(self.output)
18          }
19          out_height = (H + 2 * pad - filter_H) // stride +
    1
20          out_width = (W + 2 * pad - filter_W) // stride +
    1
21          return (self.output, out_height, out_width)
22
23      def forward(self, x, mode):
24          self.x, stride, pad = x, self.stride, self.pad
25          w, b = self.params['w'], self.params['b']
26          N, _, H, W = x.shape
27          num_filters, _, filter_height, filter_width =
    w.shape
```

```
28
29          out_height = (H + 2 * pad - filter_height) //
    stride + 1
30          out_width = (W + 2 * pad - filter_width) //
    stride + 1
31          out = np.zeros((N, num_filters, out_height,
    out_width), dtype=x.dtype)
32
33          x_pad = np.pad(x, ((0,), (0,), (pad,), (pad,)),
34                         mode='constant',
    constant_values=0)
35          for i in range(out_height):
36              for j in range(out_width):
37                  x_pad_masked = x_pad[:, :,
38                                       i * stride:i *
    stride + filter_height,
39                                       j * stride:j *
    stride + filter_width]
40                  for k in range(num_filters):
41                      out[:, k, i, j] = np.sum(
42                          x_pad_masked * w[k, :, :, :],
    axis=(1, 2, 3))
43
44          out = out + b[None, :, None, None]
45
46          return out
47
48      def backward(self, dout, mode):
49          x, w = self.x, self.params['w']
50          stride, pad = self.stride, self.pad
51          N, C, H, W = x.shape
52          num_filters, _, filter_height, filter_width =
    w.shape
53
54          dx = np.zeros_like(x)
55          x_pad = np.pad(x, ((0,), (0,), (pad,), (pad,)),
56                         mode='constant',
    constant_values=0)
57          dx_pad = np.zeros_like(x_pad)
58          dw = np.zeros_like(w)
59          db = np.sum(dout, axis=(0, 2, 3))
60
61          out_height = (H + 2 * pad - filter_height) //
    stride + 1
62          out_width = (W + 2 * pad - filter_width) //
    stride + 1
63
64          for i in range(out_height):
65              for j in range(out_width):
66                  f_w_h_start, f_w_h_end = i * stride, i *
    stride + filter_height
67                  f_w_w_start, f_w_w_end = j * stride, j *
    stride + filter_width
69                  x_pad_mask = x_pad[:, :,
70                                     f_w_h_start:
    f_w_h_end,
71                                     f_w_w_start:
    f_w_w_end]
72                  for k in range(num_filters):
```

```
72                      dw_c = x_pad_mask * (dout[:, k, i,
    j])[:, None, None, None]
73                      dw[k, :, :, :] += np.sum(dw_c,
    axis=0)
74                  for n in range(N):
75                      dx_c = w[:, :, :, :] * \
76                          (dout[n, :, i, j])[:, None, None,
    None]
77                      dx_pad[n, :, f_w_h_start:f_w_h_end,
78                             f_w_w_start:f_w_w_end] +=
    np.sum(dx_c, axis=0)
79          if self.padding == 'same':
80              dx = dx_pad[:, :, pad:-pad, pad:-pad]
81          else:
82              dx = dx_pad
83
84          grads = {
85              'w': dw,
86              'b': db,
87          }
88
89          return dx, grads
```

Source Code 5.4 Implementasi *lapisan convolution*

5.4.5 Implementasi Lapisan ReLU

```
1   import numpy as np
2   from layers.layer import Layer
3
4   class ReLU(Layer):
5       def forward(self, x, mode):
6           self.mask = x > 0
7           out = x * self.mask
8
9           return out
10
11      def backward(self, dout, mode):
12          dx = dout * self.mask
13          return dx, None
```

Source Code 5.5 Implementasi lapisan ReLU

Penjelasan Source Code 5.5, pada baris ke:

6. Membuat matriks biner *mask* untuk menhilangkan nilai *input* yang dibawah 0.

7. Mengalikan input dengan *mask*.

12. Mengalikan turunan parsial dengan *mask*.

5.4.6 Implementasi Lapisan *Max Pool*

```
1   import numpy as np
2   from layers.layer import Layer
3
4   class Pool(Layer):
5       def __init__(self, pool_type, global_pool=False,
6   filter_shape=(2, 2),
7                    stride=1):
8           super().__init__()
```

```
9           self.type = pool_type
10          self.global_pool = global_pool
11          self.stride = stride
12          self.filter_shape = filter_shape
13
14      def _get_shape(self, input_shape):
15          if self.global_pool:
16              self.filter_shape = (input_shape[1],
17  input_shape[2])
18          C, H, W = input_shape
19          filter_H, filter_W = self.filter_shape[0],
    self.filter_shape[1]
20          stride = self.stride
21          H_out = int((H - filter_H) // stride + 1)
22          W_out = int((W - filter_W) // stride + 1)
23          return (C, H_out, W_out)
24
25      def forward(self, x, mode):
26          self.x = x
27          stride = self.stride
28          f_H, f_W = self.filter_shape[0],
    self.filter_shape[1]
29          N, C, H, W = x.shape
30
31          H_out = int((H - f_H) // stride + 1)
32          W_out = int((W - f_W) // stride + 1)
33          out = np.zeros((N, C, H_out, W_out))
34          for i in range(H_out):
35              for j in range(W_out):
36                  x_masked = x[:, :, i * stride: i * stride
    +
                                 f_H, j * stride: j * stride
    + f_W]
37                  out[:, :, i, j] = np.max(x_masked,
    axis=(2, 3))
38
39          return out
40
41      def backward(self, dout, mode):
42          x, stride = self.x, self.stride
43          f_H, f_W = self.filter_shape[0],
    self.filter_shape[1]
44
45          N, C, H, W = x.shape
46          H_out = int((H - f_H) // stride + 1)
47          W_out = int((W - f_W) // stride + 1)
48          dx = np.zeros_like(x)
49
50          for i in range(H_out):
51              for j in range(W_out):
52                  w_start, w_end = i * stride, i * stride +
    f_W
53                  h_start, h_end = j * stride, j * stride +
    f_H
54                  x_masked = x[:, :, h_start: h_end,
    w_start: w_end]
55                  max_x_masked = np.max(x_masked, axis=(2,
    3))
56                  temp_binary_mask = np.asarray(
```

```
57                          x_masked == max_x_masked[:, :, None,
    None], np.int32)
58                  dx[:, :, h_start: h_end, w_start: w_end]
    += (
59                          temp_binary_mask * dout[:, :, i,
    j][:, :, None, None])
60
61          return dx, None
```

Source Code 5.6 Implementasi lapisan *max pool*

5.4.7 Implementasi Lapisan *Dropout*

```
1   import numpy as np
2   from layers.layer import Layer
3
4   class Dropout(Layer):
5       def __init__(self, keep):
6           super().__init__()
7           self.keep = keep
8
9       def forward(self, x, mode):
10          out = None
11          if mode == 'train':
12              self.mask = np.random.rand(*x.shape) <=
    self.keep
13              out = x * self.mask
14          elif mode == 'test':
15              out = x * self.keep
16
17          out = out.astype(x.dtype, copy=False)
18          return out
19
20      def backward(self, dout, mode):
21          dx = None
22          if mode == 'train':
23              dx = dout * self.mask
24          elif mode == 'test':
25              dx = dout
26          return dx, None
```

Source Code 5.7 Implementasi lapisan *dropout*

Penjelasan Source Code 5.7, pada baris ke:

12. Membuat matriks biner *mask* untuk menghilangkan nilai *input* secara acak dengan probabilitas (1-*keep*).
13. Mematikan beberapa neuron *input* dengan mengalikan dengan *mask*.
15. Mengalikan input dengan nilai probabilitasi *keep* untuk menjaga ukuran matriks input sama pada saat pelatihan dan pengujian.
16. Dalam mode pengujian turunan parsial dikecilkan dengan mengalikan probabilitas *keep*.

5.4.8 Implementasi Lapisan *Merge*

```
1   import numpy as np
2   from layers.layer import Layer
3
4
5   class Merge(Layer):
6       def __init__(self, axis):
7           super().__init__()
8           self.axis = axis
9
10      def _check_input(self, x):
11          if type(x) is not tuple:
12              raise Exception('Input to this layer must be
    a tuple')
13
14      def _get_shape(self, input_shape):
15          self._check_input(input_shape)
16          axis = self.axis - 1
17          self.num_input = len(input_shape)
18          out = None
19          for shape in input_shape:
20              if out is None:
21                  out = list(shape)
22              else:
23                  out[axis] += shape[axis]
24          out = tuple(out)
25          return out
26
27      def forward(self, x, mode):
28          self._check_input(x)
29          out = np.concatenate(tuple(x), axis=self.axis)
30          return out
31
32      def backward(self, dout, mode):
33          dx = np.split(dout, self.num_input,
    axis=self.axis)
34          return tuple(dx), None
```

Source Code 5.8 Implementasi lapisan *merge*

Penjelasan Source Code 5.8, pada baris kode:

19-23. Menghitung ukuran kedua input setelah digabungkan.

29. Menggabungkan dua matriks pada sumbu yang telah ditentukan.

33. Memisah turunan parsial menjadi dua pada sumbu yang telah ditentukan.

5.4.9 Implementasi Lapisan *Global Average Pool*

```
1   import numpy as np
2   from layers.layer import Layer
3
4   class GlobalAveragePool(Layer):
5       def _get_shape(self, input_shape):
6           self.input_shape = input_shape
7
8       def forward(self, x, mode):
9           N, C, _, _ = x.shape
```

```
10          self.N = N
11          out = x.mean(axis=(2, 3))
12          out.reshape(N, C)
13          return out
14
15      def backward(self, dout, mode):
16          N = self.N
17          C, H, W = self.input_shape
18          avg = 1 / (H * W)
19          dx = np.full((N, C, H, W), avg)
20          for i in range(N):
21              for j in range(C):
22                  dx[i, j, :, :] *= dout[i, j]
23          return dx, None
```

Source Code 5.9 Implementasi lapisan *global average pool*

Penjelasan Source Code 5.9, pada baris ke:

11. Menghitung *output* dengan menghitung rata-rata *input* pada sumbu ke 2&3 (tinggi dan lebar *input*).

18. Menghitung pembagi turunan parsial secara merata.

20-22. Membagi turunan parsial dengan jumlah *input* pada setiap activation map.

BAB 6 PENGUJIAN DAN ANALISIS

Pada bagian ini akan dibahas tekait proses pengujian terhadap sistem "Klon Perilaku Menggunakan Jaringan Saraf Tiruan Konvolusional di Dalam *Game SuperTuxKart*" yang telah diimplementasikan. Pada proses pengujian dilakukan berdasarkan pada perancangan pengujian yang telah dirancang sebelumnya. Pengujian sistem meliputi pengujian pengaruh jumlah *epoch,* pengaruh *learning rate,* pengaruh parameter *squeeze* dan *expand*. Dalam semua kondisi setiap pengujian dilakukan sebanyak lima kali running sistem, kemudian diambil nilai rata-rata evalausinya, evaluasi sistem menggunakan *error* dan akurasi. Dilakukannya proses pengujian untuk mengetahui hasil evaluasi sistem terhadap implementasi metode yang telah digunakan.

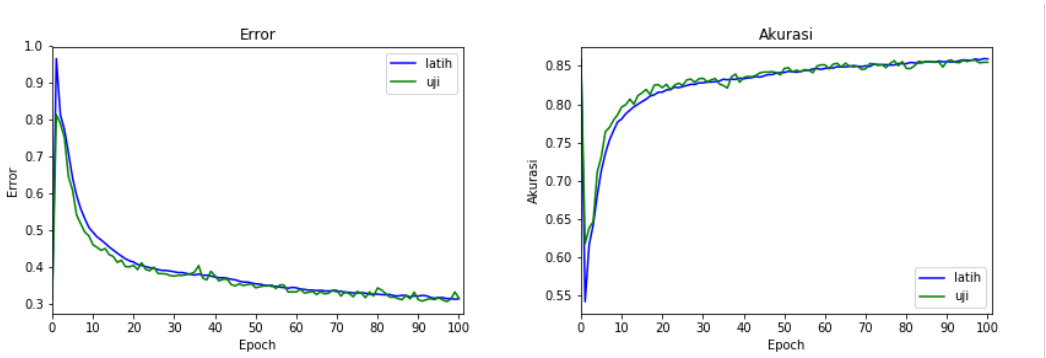
6.1 Pengujian dan Analisis Pengaruh Jumlah *Epoch*

Pada proses pengujian pertama dilakukan untuk mengetahui pengaruh jumlah *epoch* terhadap hasil evaluasi sistem yang telah diimplementasikan. Dalam pengujian ini dilakukan dengan jumlah *epoch* 5, 10, 20 dan 30. Nilai pengujian ini digunakan karena jika pengulangan pada *epoch* terlalu banyak atau terlalu sedikit, perubahan yang akan diamati tidak akan signifikan. Pengujian ini bertujuan untuk mendapatkan jumlah *epoch* terkecil dengan nilai evaluasi tertinggi, sehingga akan memudahkan pada proses komputasi. Setiap jumlah *epoch* akan dilakukan pengujian terhadap data uji sebanyak lima kali dan nilai *learning rate* bernilai tetap yakni 0,0001 dan dan *beta* 1 bernilai 0,5. Dalam pengujian ini didapatkan nilai evaluasi *error* dan akurasi yang berbeda-beda. Untuk nilai hasil evaluasi pengujian pengaruh jumlah *epoch* dapat dilihat pada Tabel 6.1.

Tabel 6.1 Hasil pengujian pengaruh jumlah *epoch*

Jumlah *Epoch*	**Nilai *Learning Rate* (α)**	**Hasil Evaluasi**			
		error		**Akurasi (%)**	
		latih	**uji**	**latih**	**uji**
10	0,0001	0,4951	0,4613	78,08	79,65
25		0,3971	0,4005	82,32	82,46
50		0,3548	0,3445	84,15	84,68
100		0,3146	0,3164	85,93	85,49

Grafik hubungan antara pengaruh jumlah *epoch* dengan hasil akurasi sistem telah disajikan pada grafik pengaruh jumlah *epoch* pada Gambar 6.1.



Gambar 6.1 Pengaruh jumlah *epoch*

Berdasarkan Tabel 6.1 dan Gambar 6.1 akurasi sudah mencapai 80% pada *epoch* ke-12 dengan *error* 0,4742 dan mencapai puncak akurasi terbaik pada epoch ke-97 yaitu sekitar 85,73% *error* juga mencapai titik terendahnya pada *epoch* ke-97 yaitu 0,3151.

Hasil yang diperoleh pada grafik pengaruh jumlah *epoch* menunjukan bahwa perubahan jumlah *epoch* berpengaruh pada nilai evaluasi yang dihasilkan. Perubahan pada jaringan saraf tiruan sangat cepat sampai pada *epoch* ke-20. Perubahan setalah *epoch* ke-20 sudah tidak signifikan, sehingga meningkatkan jumlah *epoch* tidak akan mengubah meningkatkan akurasi dan dan menurunkan *error* secara signifikan. Berdasarkan Tabel 6.1 dan Gambar 6.1 dapat disimpulkan bahwa jumlah *epoch* dengan nilai 100 adalah nilai yang optimal.

Hasil di atas menujukkan bahwa pada pelatihan dengan nilai *epoch* terbaik adalah 100. Karena nilai *error* pada *epoch* ke-100 lebih rendah daripada *epoch-epoch* sebelumnya dengan akurasi yang lebih tinggi daripada *epoch-epoch* sebelumnya.

6.2 Pengujian dan Analisis Pengaruh *Learning Rate*

Pada pengujian kedua ini bertujuan untuk mengetahui pengaruh perubahan dari *learning rate* terhadap nilai evaluasi sistem yang dihasilkan. Nilai *learning rate* yang digunakan dalam pengujian ini adalah 0,01, 0,001 dan 0,0001. Penentuan nilai *learning rate* ini diambil karena *range* nilai yang diperbolehkan adalah diantara 0 dan 1, selain itu apabila nilai *learning rate* yang digunakan terlalu kecil maka hasil evaluasi sistem yang didapatkan nanti tidak akan terlihat signifikan untuk dilakukan pengamatan. Setiap nilai *learning rate* akan dilakukan pelatihan dengan parameter *beta* 1 bernilai 0,7 dan jumlah *epoch* maksimal bernilai 30. Hasil dari pengujian pengaruh *learning rate* bisa dilihat pada Tabel 6.2.

Tabel 6.2 Hasil Pengujian Pengaruh *Learning Rate*

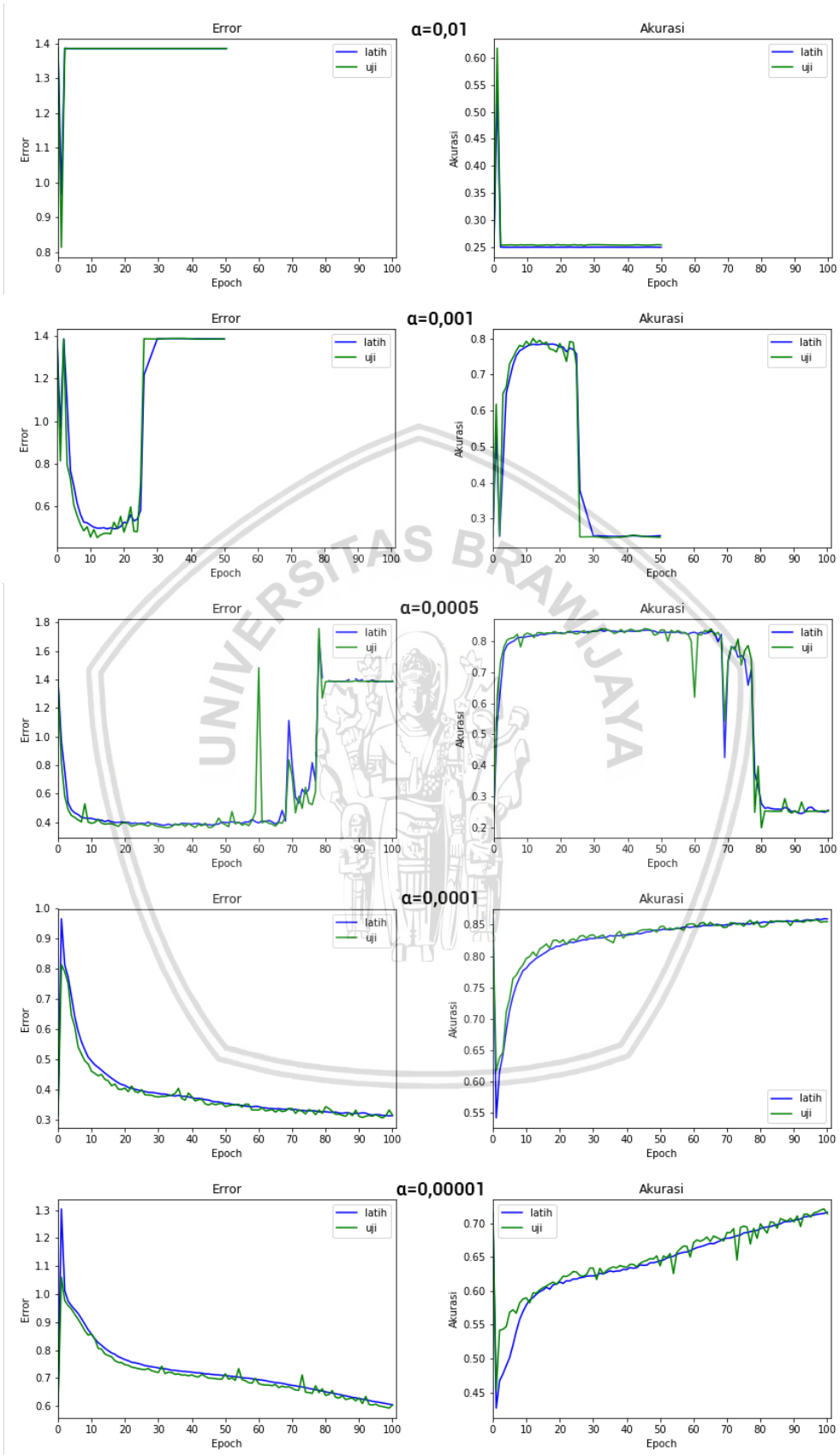
Nilai *Learning Rate* (α)	*Epoch Terbaik*	Hasil Evaluasi Terbaik			
		error		Akurasi (%)	
		latih	uji	latih	uji
0,01	1	0,9654	0,8136	54,21	61,77
0,001	12	0,4961	0,4521	78,44	80,08
0,0005	29	0,3864	0,3611	83,02	83,85
0,0001	97	0,3151	0,307	85,92	85,74
0,00001	99	0,607	0,5933	71,47	72,14

Grafik hubungan antara pengaruh *lerning rate* dengan hasil akurasi sistem telah disajikan pada grafik pengaruh jumlah *epoch* pada Gambar 6.2.

Berdasarkan grafik pada Gambar 6.2, hasil evaluasi sistem pada pengujian ini berbeda nilai *error* dan akurasinya ketika nilai *learning rate* berubah-ubah. Ketika *learning rate* adalah 0,01 menyebabkan sebagian besar *neuron* pada lapisan *convolution* mati sehingga gradien tidak dapat mengalir pada lapisan *ReLU* ketika propagasi mundur. Oleh karena itu maka pelatihan dihentikan pada *epoch* ke-50.

Pada *learning rate* 0,001 *error* semakin meningkat pada *epoch* ke-2 dan kembali turun dengan cepat dan stabil pada epoch ke-7. Titik *error* terendah 0,4521 pada *epoch* ke-12 dengan akurasi 80,08%. Karena *learning rate* yang masih terlalu besar, proses modifikasi bobot *neuron* terlalu besar hingga menyebabkan bobot neuron yang baru menjadi negatif atau terlalu besar sehingga gradien menjadi 0 dan pelatihan dihentikan pada *epoch* ke-50. Pada *learning rate* 0,0005 *error* berkurang lebih lambat daripada *learning rate* sebelumnya tapi mencapai titik *error* yang lebih rendah pada *epoch* ke-29 dengan nilai 0,3611 dan akurasi 83,85%. Saat *learning rate* adalah 0,0001 *error* terus turun dan akurasi naik meningkat hingga akhir *epoch*. *Error* mencapai titik terendahnya pada *epoch* ke-97 dengan nilai 0,307 dan akurasi 85,74%. *Learning rate* ini cukup baik karena dapat mencapai titik *error* terendah dibandingkan *learning rate* sebelumnya. Saat learning rate adalah 0,00001 *error* menurun sangat lambat hingga akhir *epoch* dan mencapai titik terendahnya pada *epoch* ke-99 dengan nilai 0,607 dan akurasi 72,14% pada data uji. *Learning rate* ini terlalu kecil dan bergerak sangat lambat menuju titik minimum lokal. Walaupun pada akhirnya *error* dapat diperkecil dan akurasi semakin membaik, tapi membutuhkan waktu yang lebih lama untuk mencapainya.

Pada hasil yang diperoleh pada pengujian pengaruh *learning rate* menunjukan bahwa perubahan nilai *learning rate* memberikan yang signifikan pada nilai *error* dan akurasi yang dihasilkan. Nilai *learning rate* yang akan diambil adalah dengan memilih nilai *error* terendah, akurasi terbaik dan *epoch* paling sedikit. Kemudian disimpulkan bahwa *learning rate* 0,0001 adalah nilai paling optimal dengan akurasi 85,74%, *error* 0,3611 dan pada *epoch* ke-97.



Gambar 6.2 Pengujian pengaruh learning rate

6.3 Pengujian dan Analisis Pengaruh *Momentum*

Pada pengujian kedua ini bertujuan untuk mengetahui pengaruh perubahan dari *beta 1* terhadap nilai evaluasi sistem yang dihasilkan. *Beta* 1 berperan sebagai pereduksi momentum atau gaya gesek dalam fisika. Nilai *beta* 1 yang digunakan dalam pengujian ini adalah 0,3, 0,5 dan 0,7. Penentuan nilai *beta 1* ini diambil kerena *range* nilai yang diperbolehkan adalah diantara 0 dan 1, selain itu apabila nilai *beta 1* yang digunakan terlalu kecil maka efek inersia semakin kecil sehingga sangat terpengaruh pada perubahan gradien bidang optimisasi. Nilai *beta* 1 yang kecil juga dapat menyebabkan konvergensi dini pada bidang optimisasi yang berbukit-bukit. Jika nilai *beta 1* semakin besar maka lebih lama berhenti pada titik minimum lokal. Setiap nilai *beta 1* akan dilakukan pelatihan dengan jumlah *epoch* 100 yang didapatkan dari pengujian sebelumnya. Hasil dari pengujian pengaruh *momentum* bisa dilihat pada Tabel 6.3.

Tabel 6.3 Hasil Pengujian Pengaruh *Momentum*

Nilai *Beta 1* (β_1)	*Epoch* terbaik	Hasil Evaluasi			
		error		Akurasi (%)	
		latih	uji	latih	uji
0,3	100	0,319	0,3033	85,87	86,72
0,5	97	0,3151	0,307	85,92	85,74
0,7	97	0,3827	0,3746	83,06	83,46

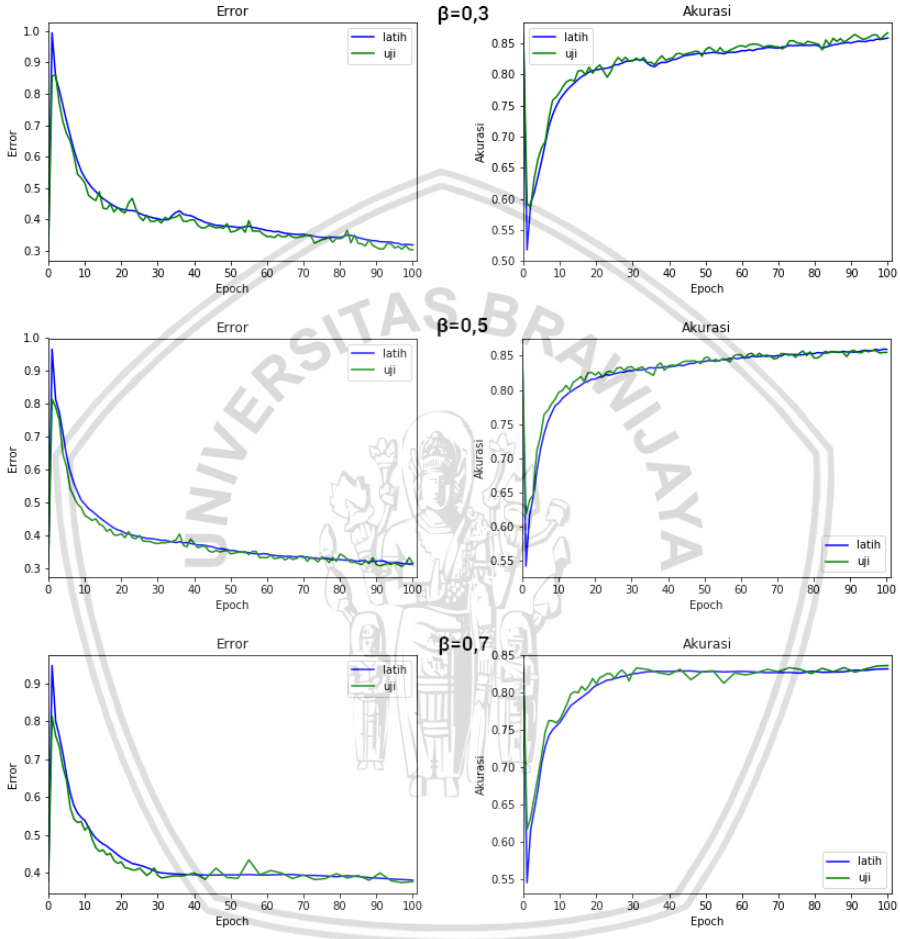
Grafik hubungan antara pengaruh *momentum* dengan hasil akurasi sistem telah disajikan pada grafik pengaruh *momentum* pada Gambar 6.3.

Berdasarkan grafik pada Gambar 6.3, terlihat bahwa hasil evaluasi sistem pada pengujian ini berbeda nilai *error* dan akurasinya ketika nilai *beta 1* berubah-ubah. Ketika *beta* 1 bernilai 0,3 titik *error* terendah ada pada *epoch* ke-100 yaitu 0,319 pada data latih dan 0,3033 pada data uji dengan akurasi 85,87% pada data latih dan 86,72% pada data uji. Kecilnya nilai *beta 1* mengakibatkan inersia semakin kecil yang berakibat arah optimisasi menjadi cepat berubah ketika gradien bidang berubah. Perubahan arah optimisasi menyebabkan proses optimisasi menuju titik minimum menjadi terganggu dan lebih lama tetapi *error* dapat terus turun ketika mendekati titik minimum lokal karena tidak terlalu terpengaruh *momentum*.

Ketika *beta 1* adalah 0,5 titik *error* terendah terjadi pada *epoch* ke-97 yaitu 0,3151 pada data latih dan 0,307 pada data uji dengan akurasi 85,92% pada data latih dan 85,74% pada data uji. Dibandingkan pengujian sebelumnya, *error* dapat turun lebih cepat pada awal pelatihan dan lebih lambat ketika mendekati titik minimum lokal.

Ketika *beta 1* adalah 0,7 titik *error* terendah terjadi pada *epoch* ke-97 yaitu 0,387 pada data latih dan 0,3746 pada data uji dengan akurasi 83,46% pada data latih dan 83,46% pada data uji. Nilai *beta* 1 yang lebih besar mengakibatkan perubahan arah optimisasi yang lebih sedikit walaupun menjadi sangat lambat saat mendekati titik minimum lokal.

Pada hasil yang diperoleh pada pengujian pengaruh *momentum* menunjukan bahwa perubahan nilai *beta 1* memberikan yang tidak terlalu signifikan pada nilai *error* dan akurasi yang dihasilkan. Nilai *beta 1* yang akan diambil adalah dengan memilih nilai *error* terendah, akurasi terbaik dan *epoch* paling sedikit. Kemudian disimpulkan bahwa *beta 1* 0,3 adalah nilai paling optimal dengan akurasi 86,72%, *error* 0,3033 dan pada *epoch* ke-100.



Gambar 6.3 Pengujian pengaruh momentum

BAB 7 PENUTUP

Bagian ini memuat kesimpulan dan saran terhadap skripsi. Kesimpulan dan saran disajikan secara terpisah, dengan penjelasan sebagai berikut:

7.1 Kesimpulan

Berdasarkan hasil penelitian ini, dapat disimpulkan beberapa hal mengenai klon perilaku pemain menggunakan jaringan saraf tiruan dalam *game supertuxkart*.

1. Klon perilaku dapat dilakukan dengan mengggunakan jaringan saraf tiruan. Data yang diambil adalah tangkapan tampilan layar beserta tombol *keyboard* yang ditekan. Selanjutnya adalah merancang algoritme untuk melakukan pelatihan dan prediksi. Setiap lapisan juga perlu dirancang untuk dapat melakukan propagasi maju dan mundur. Setelah rancangan telah diimplementasikan. Mulai proses pelatihan dengan *learning rate* yang optimal. Setelah jaringan saraf tiruan terlatih, implementasi program untuk menggunakan jaringan saraf tiruan agar dapat bermain game *SuperTuxKart*.
2. Akurasi beragam dan sangat bergantung pada parameter model seperti *learning rate*, jumlah *epoch* dan *momentum*. Berdasarkan hasil pengujian dari penelitian ini, *learning rate* yang optimal adalah 0,0001, jumlah *epoch* adalah 100 dan momentum adalah 0,3.

7.2 Saran

Berdasarkan hasil penelitian ini, beberapa saran mengenai klon perilaku pemain menggunakan jaringan saraf tiruan dalam *game supertuxkart*.

1. Melakukan pengujian pada setiap parameter dari model untuk menentukan parameter seperti *learning rate*, jumlah *epoch* dan *momentum* yang optimal.
2. Menentukan parameter yang dinamis di dalam *game* berdasarkan spesifikasi komputer pemain.
3. Menggunakan *dataset* yang seimbang untuk menghindari model menjadi *overfit* dan memiliki prediksi yang dominan.

DAFTAR PUSTAKA

Abbott, R. G., 2007. Behavioral Cloning for Simulator Validation. Dalam: *RoboCup 2007: Robot Soccer World Cup XI.* s.l.:s.n., pp. 329-336.

Baldi, P. & Sadowski, P., 2014. The dropout learning algorithm.

Bojarski, M. et al., 2016. End to End Learning for Self-Driving Cars.

Botev, A., Lever, G. & Barber, D., 2016. Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent.

Chen, Z. & Yi, D., 2017. The Game Imitation: Deep Supervised Convolutional Networks for Quick Video Game AI.

Dreyfus, S., 1962. The numerical solution of variational problems.

Duchi, J., Hazan, E. & Singer, Y., 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research,* 12(Jul).

Fukushima, K., 1980. Biological Cybernetics Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.

Green, C., 2016. *Cross Entropy.* [Online] Available at: http://heliosphan.org/cross-entropy.html [Diakses 14 2 2018].

He, K., Zhang, X., Ren, S. & Sun, J., 2015. Deep Residual Learning for Image Recognition.

Hinton, G. E. et al., 2012. Improving neural networks by preventing co-adaptation of feature detectors.

Humphrey, E. J., Bello, J. P. & Lecun, Y., 2012. Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning In Music Informatics.

Iandola, F. N. et al., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and.

Kingma, D. P. & Ba, J., 2014. Adam: A Method for Stochastic Optimization.

Lin, M., Chen, Q. & Yan, S., 2013. Network In Network.

Lueangrueangroj, S. & Kotrajaras, V., t.thn. Real-Time Imitation Based Learning for Commercial Fighting Games.

Nair, V. & Hinton, G. E., 2009. Rectified Linear Units Improve Restricted Boltzmann Machines.

Nielsen, M. A., 2015. Improving the way neural networks learn. Dalam: *Neural Networks and Deep Learning.* s.l.:Determination Press.

OpenAI, 2017. *Dota 2.* [Online] Available at: https://blog.openai.com/dota-2/ [Diakses 18 12 2017].

Puzenat, D. & Verlut, I., 2010. Behavior Analysis through Games Using Artificial Neural Networks.

Rall, L. B., 2006. Perspectives on Automatic Differentiation: Past, Present, and Future?. Dalam: M. Bücker, et al. penyunt. *Automatic Differentiation: Applications, Theory, and Implementations.* Berlin/Heidelberg: Springer-Verlag, pp. 1-14.

Robinel, A. & Puzenat, D., 2014. Alcohol consumption detection through behavioural analysis using intelligent systems.

Ross, S. & Bagnell, D., 2010. Efficient Reductions for Imitation Learning.

Ruder, S., 2016. An overview of gradient descent optimization algorithms.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J., 1986. Learning Internal Representations by Error Propagation.

Schmidhuber, J., 2014. Deep Learning in Neural Networks: An Overview.

Silver, D. et al., 2016. Mastering the game of Go with deep neural networks and tree search. *Nature,* Volume 529.

SuperTuxKart Team, 2016. *SuperTuxKart.* [Online] Available at: https://supertuxkart.net/FAQ [Diakses 20 12 2017].

Weng, J., Ahuja, N. & Huang, T., 1992. *Cresceptron: a self-organizing neural network which grows adaptively.* s.l., s.n.

Werbos, P. J., 2006. Backwards Differentiation in AD and Neural Nets: Past Links and New Opportunities. Dalam: M. Bücker, et al. penyunt. *Automatic Differentiation: Applications, Theory, and Implementations.* Berlin/Heidelberg: Springer-Verlag, pp. 15-34.